

El mundo de la COMPUTACION

Curso teórico-práctico

3

OCEANO


COLEGIO DE E.G.B.
"LOS BOSCOS"
LOGROÑO

El mundo de la COMPUTACION

Curso teórico-práctico

3

OCEANO

Es una obra del

GRUPO EDITORIAL OCEANO

Presidente
José Lluis Monreal

Director General
José M.ª Martí

Director General de Publicaciones
Carlos Gispert

Paseo de Gracia, 24
Teléfonos: 317 45 08*
Télex: 51735 exit e - Fax: 317 97 01
08007 Barcelona (España)

Impreso en España - Printed in Spain

ISBN: 84-7764-189-7 (Obra completa)
ISBN: 84-7764-192-7 (Volumen 3)

Depósito legal: CO-877-92 (Oc)
Impreso y encuadernado por:
GRAFICROMO, S.A. Córdoba (España)

volumen

3



Software. Conceptos básicos
en la programación de computadoras
Lenguajes de programación. El lenguaje BASIC
El futuro tecnológico-científico y las computadoras

Software

Para su buen funcionamiento, toda computadora necesita de dos elementos indispensables: el *hardware* y el *software*. El hardware está formado por todas las partes visibles de la computadora, como la pantalla, el teclado, la unidad central, etc. Por otro lado, el software es un elemento totalmente intangible, pero que sin él la computadora nunca podría funcionar. Veamos un ejemplo.

Tenemos una computadora con la que queremos realizar un determinado trabajo. Este trabajo comprende toda una serie de procesos intermedios necesarios para llegar al final de la tarea con éxito. De algún modo hay que informar a la computadora de cómo realizar estos procesos intermedios para que pueda ejecutarlos. *Estas órdenes que se dan a la computadora, siguiendo una terminología determinada y nunca de manera arbitraria, conforman el elemento software.*

Descripción del software

Se han desarrollado diversas técnicas para analizar los trabajos que realiza una computadora, así como un conjunto de símbolos y palabras, producto del análisis efectuado, creado expresamente para ordenar de modo racional los pasos que ha de dar la computadora para realizar estos trabajos. Este conjunto de órdenes constituye lo que comúnmente se llama *programa*. A cada una de las órdenes que componen este programa se la denomina *instrucción* o *sentencia*. El término general que se utiliza para indicar que se está ordenando algo a una computadora es el de *programación*; al conjunto de símbolos o palabras que se utilizan para representar estas órdenes se le llama *lenguaje de programación*.

Este nivel de software que hemos descrito es el más conocido y aplicado por las personas que realizan la programación de las computadoras. Los que utilizan el software pueden ser simples usuarios o técnicos especializados que, en cualquier caso, reciben el nombre de *programadores*. Con todo, dentro del software existe otro nivel mucho más complejo y especializado que se encarga de efectuar el enlace entre los programas y los elementos hardware.

El software se divide en dos apartados:

- *Software de base.*
- *Software aplicativo.*

El proceso de traducción

Los intérpretes

Cualquier lenguaje de programación se puede traducir mediante un intérprete construido expresamente para este lenguaje; de ahí que no exista un intérprete único para todos los lenguajes. Sin embargo, el único lenguaje para el que se ha creado un intérprete es el lenguaje BASIC. El trabajo de un intérprete, como su nombre indica, es el de traducir las instrucciones del lenguaje de programación a lenguaje máquina, de tal manera que la C.P.U. pueda ejecutar dichas instrucciones en lenguaje máquina. Para ello, tanto el programa usuario como el programa intérprete se encontrarán en memoria; esto significa que la traducción será simultánea.

Los compiladores

Los compiladores traducen las sentencias o instrucciones del lenguaje de programación y las convierten en un conjunto de instrucciones en lenguaje máquina directamente ejecutables por la computadora.
El proceso de traducción con compilador no se realiza simultáneamente, como en el caso de los intérpretes, sino que se lleva a cabo en un proceso aparte.
El compilador toma el conjunto de instrucciones del lenguaje de programación, que se denomina programa fuente, *como datos de entrada y las convierte en instrucciones de lenguaje máquina cuyo conjunto pasa a denominarse* programa objeto. *De esta manera, cuando la C.P.U. ejecuta el programa ya no necesita un traductor, porque ella misma ejecuta directamente las instrucciones del programa ya traducido.*
Existen compiladores para muchos lenguajes, por ejemplo, para COBOL, FORTRAN, PASCAL, BASIC.

SOFTWARE DE BASE

El software de base está formado por toda una serie de programas que sirven de enlace entre los programas escritos por un programador con el fin de realizar un determinado trabajo y los elementos hardware de la computadora. Por ejemplo: un programador puede ordenar a una computadora que imprima una frase en la impresora. Componer esta orden resulta muy fácil cualquiera que sea el lenguaje de programación que se utilice, pero no es tan sencilla su ejecución: «alguien» ha de estar informado de la necesidad de ejecución de esa orden, «alguien» tiene que traducirla a un lenguaje que la computadora pueda comprender; finalmente, «alguien» tiene que controlar el buen funcionamiento de la operación. Este «alguien» es el *software de base*, que está formado fundamentalmente por los elementos que se citan a continuación:

- Sistema operativo.
- Los traductores:
  - los intérpretes;
  - los compiladores.
- El ensamblador.
- Los programas de utilidad.

El *sistema operativo* es el elemento principal del software de base y de él hablaremos en un capítulo aparte.

Los traductores

La computadora sólo puede ejecutar instrucciones en un lenguaje al que normalmente se denomina *lenguaje máquina*. Por ello, cualquier lenguaje de programación que no sea lenguaje máquina necesitará un proceso de traducción. Este proceso lo llevan a cabo los intérpretes y los compiladores.

El ensamblador

El lenguaje máquina es el que la computadora puede ejecutar directamente; sin embargo, este lenguaje no puede denominarse como tal a causa de que no está formado por símbolos o signos, como cualquier otro lenguaje, sino que está compuesto por cantidades numéricas expresadas en base 16 o hexadecimal. De ello se deduce que programar directamente en lenguaje máquina resulta muy complicado. Para evitar esta dificultad existe el lenguaje ensamblador, que es un lenguaje de programación muy cercano al lenguaje máquina. Esta similitud hace que el proceso de traducción del lenguaje ensamblador al lenguaje máquina se realice mediante muy pocos pasos, ya que sus sentencias guardan, en realidad, la estructura sintáctica del lenguaje máquina.

Aplicaciones estandarizadas y a medida

Pensemos en un colectivo sujeto a las mismas reglamentaciones, como es el caso de los farmacéuticos. Se puede construir una aplicación que resuelva sus problemas de almacenamiento de medicamentos, esto es, que controle todos los específicos, que informe de cuántos frascos de cada preparación se tienen en el almacén o de su consumición, para efectuar los pedidos necesarios al laboratorio proveedor, que dé a conocer en el momento requerido el valor total de los productos almacenados, etc. La aplicación que se construya para tal fin servirá a todos los farmacéuticos por igual; de ahí su nombre: estandarizada.
Si un farmacéutico quiere solucionar mediante la computadora un caso específico de su negocio, se le diseñará una aplicación exclusivamente para él, o sea, se le hará a la medida *de su problema. Cada vez se construyen menos aplicaciones a medida, ya que se acostumbran a retocar lo programas de las aplicaciones estandarizadas que no contemplen las facetas que el usuario necesite.*



Control de existencias en una farmacia con ayuda de una computadora.

Dado un programa escrito en ensamblador, para obtener el mismo programa en lenguaje máquina será preciso un proceso de traducción.

La función del ensamblador es muy parecida a la del compilador, sólo que éste traduce lenguajes de alto nivel, simbólicos, mientras que aquél traduce los de bajo nivel, cercanos al lenguaje máquina.

Los programas de utilidad

Cuando se trabaja con una computadora, se realizan una serie de trabajos repetitivos, como clasificación de ficheros, copia de los mismos o de programas de una unidad de almacenamiento en otra (de un disco a una cinta o de un disco a otro), etcétera.

Para facilitar estas tareas a los usuarios y para evitar a los programadores o a los mismos usuarios el trabajo de preparar los programas que las lleven a cabo, se distribuyen, normalmente, junto con el software de base que va incluido en la computadora, unos programas convencionales que se denominan por esta razón programas de utilidad.

SOFTWARE APLICATIVO

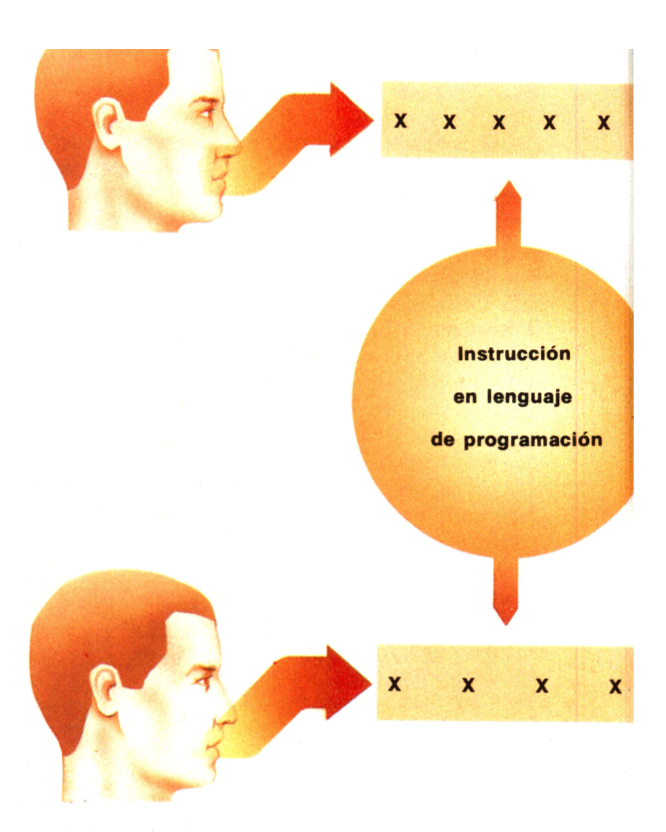
Recibe el nombre de software aplicativo todo el conjunto de programas escritos para resolver unos problemas específicos. Estos problemas no derivan de disfunciones internas de la computadora, sino que es el usuario quien los plantea y pretende su resolución mediante el uso de la computadora.

Supongamos un empresario que cada fin de año tiene problemas para dejar su contabilidad bien cerrada, sin ningún error y en el tiempo justo; y para resolver este asunto decide valerse de una computadora. Este empresario no solucionará su problema si no dispone de toda una serie de programas, que la computadora ejecutará, hechos a medida para que pueda resolver su problema de contabilidad.

Al conjunto de programas escritos para resolver un problema determinado se le llama *aplicación*. Así podemos hablar de aplicación de contabilidad, aplicación de nóminas, aplicación de almacenes, etcétera.

Cada programa que compone una aplicación sirve para la resolución de una de su partes o de un problema determinado. Así por ejemplo, la aplicación de contabilidad contendrá en primer lugar un programa para construir el plan contable, en segundo, otro para efectuar el diario contable y así sucesivamente.

Al construir todos los programas de una aplicación, puede ocurrir que éstos sirvan para resolver un problema determinado de un usuario o que se utilicen para resolver el problema



El diagrama pone de manifiesto la diferencia existente entre un intérprete y un compilador. El intérprete traduce de forma simultánea a lenguaje máquina las órdenes de ejecución dadas en lenguaje de programación. El compilador, por su parte, no es un traductor simultáneo, sino que recoge las órdenes en lenguaje de programación y se las comunica a la C.P.U. en el lenguaje que ésta puede entender: en lenguaje máquina.

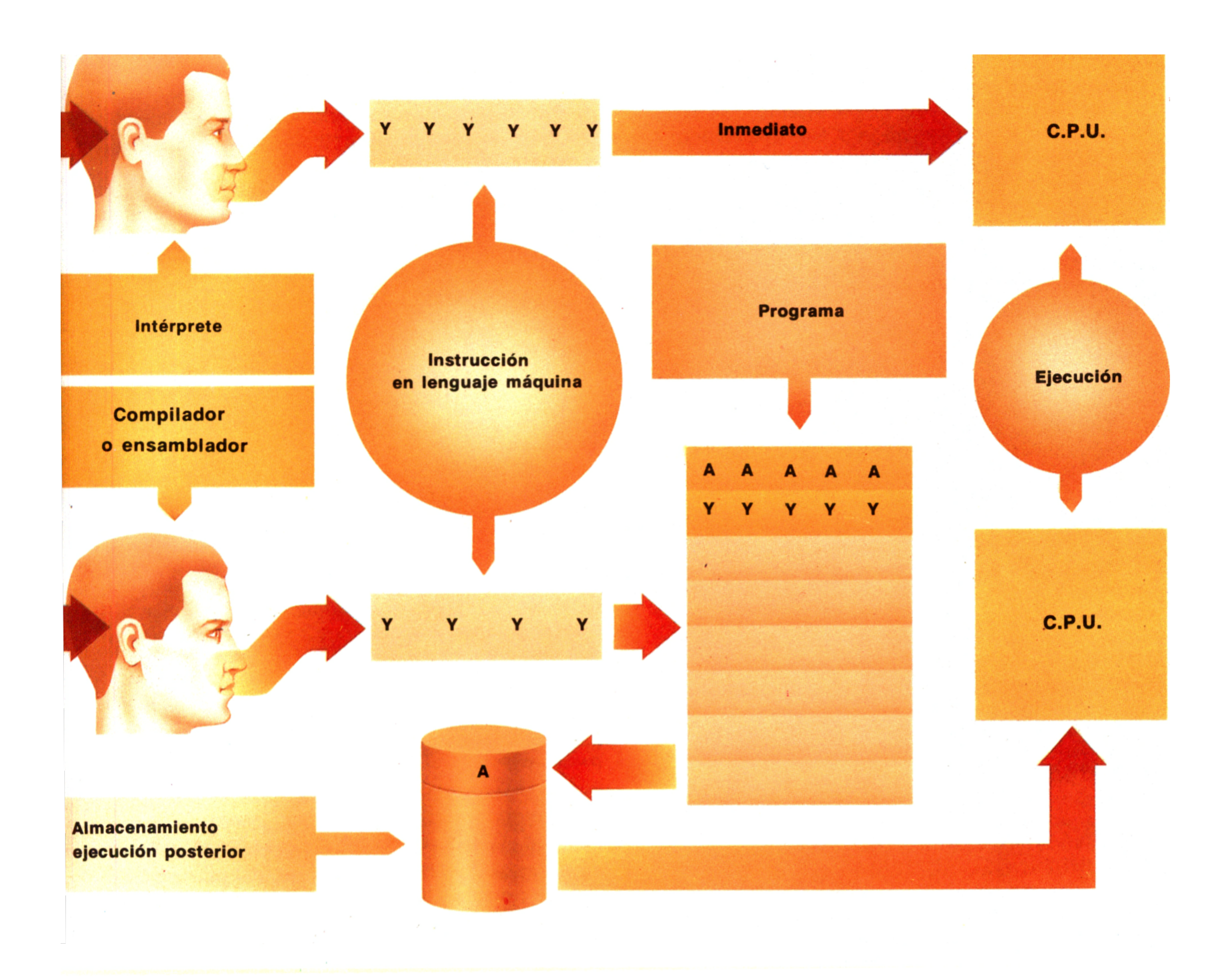
de muchos usuarios. Así podemos hablar de dos tipos de aplicaciones:

- aplicaciones estandarizadas;
- aplicaciones a medida.

EL SISTEMA OPERATIVO

Un sistema operativo consta de una serie de programas que controlan todas las actividades que la computadora realiza. Su función, por lo tanto, consiste en controlar el trabajo que la computadora efectúa. Por ejemplo, el sistema operativo controla la posibilidad o imposibilidad de ejecución de un programa; si la computadora tiene todos los recursos necesarios para llevar a término su trabajo (memoria, tiempo de C.P.U., etcétera).

Para entender mejor las funciones del sistema operativo, podemos comparar la tarea que éste realiza con el trabajo de un recepcionista de una gran compañía. El recepcionista atiende



las visitas y las dirige a la persona apropiada para que resuelva sus problemas, avisa al personal requerido por las visitas y, en caso de que no estuviera libre la persona solicitada, haría esperar al visitante. Así, el sistema operativo controla la entrada de programas (visitantes) en la memoria, permite el acceso del programa a la zona de memoria requerida (a la cinta, al fichero de disco, a la impresora) y abre paso o no a la entrada de más programas en la memoria.

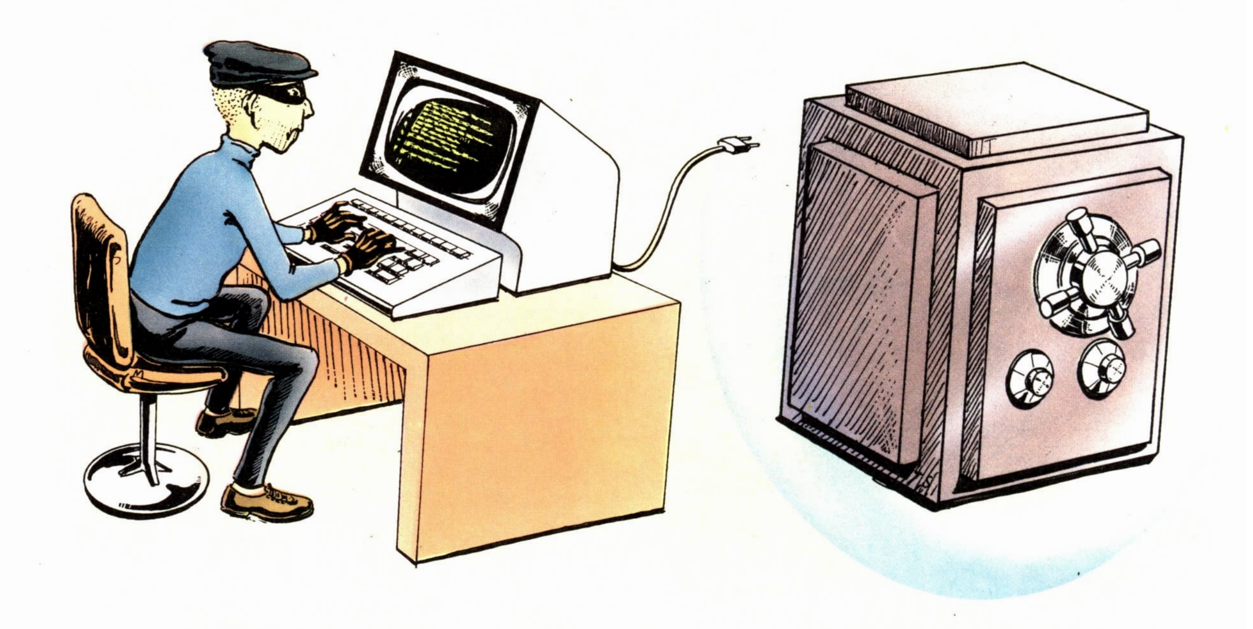
El supervisor y las funciones principales de un sistema operativo

Todas las actividades de un sistema operativo son de control, por lo que todas sus funciones están relacionadas con la inspección de todos los procesos que se efectúan en una computadora.

Antes de pasar a la descripción de cada una de las funciones del sistema operativo, conviene hacer hincapié en una de sus partes: la que reviste mayor importancia y a la que comúnmente se la denomina *supervisor* o *monitor*.

El supervisor es un programa que se activa automáticamente en el mismo momento en que la computadora se pone en marcha y que permanece siempre en memoria hasta que la computadora se desconecta. El supervisor realiza funciones de enlace entre los programas del usuario y las rutinas del sistema operativo. Cuando se está realizando cualquier trabajo con la computadora y se necesita la ejecución de algún programa o rutina del sistema operativo, el control pasa al supervisor, el cual conecta con dicho programa o rutina y le cede dicho control hasta terminada su ejecución, después de la cual éste vuelve al supervisor.

Se puede comparar el trabajo del supervisor con las funciones que realiza el moderador de un coloquio que va cediendo la palabra ordenadamente a los individuos que la solicitan, pero que, después de cada una de las intervenciones, el control de éstas vuelve siempre al moderador.



A continuación se describen las funciones de un sistema operativo correspondiente a grandes computadoras.

Al igual que las cajas de caudales, las computadoras pueden estar protegidas por una clave sólo conocida por unas pocas personas.

ASIGNACIÓN DE RECURSOS DEL SISTEMA

El sistema operativo se encarga de proporcionar el espacio de disco, cinta, etc., y el espacio de memoria necesario para cada uno de los programas que se ejecutan en una computadora. El espacio de disco o cinta, una vez asignado, ya no se libera; sin embargo, el espacio de memoria, una vez terminada la ejecución del programa que lo ocupa, queda libre gracias al sistema operativo, el cual lo podrá asignar a otro programa.

MONITORIZACIÓN

El sistema operativo efectúa actividades de monitorización tales como finalizar la ejecución de un programa cuando se ha producido un error o se ha superado el tiempo de ejecución o el espacio de memoria que tenía asignado. Al mismo tiempo que finaliza la ejecución del programa, el sistema operativo envía un mensaje a la pantalla del terminal o de la computadora desde donde se ordenó la ejecución del programa, informando al operador o usuario que se ha interrumpido la ejecución del programa y especificándole la causa de la interrupción.

Otra actividad de monitorización es el control del nivel de seguridad de los usuarios que trabajan con la computadora, de tal manera que no permite el acceso a la información protegida si no se presenta la autorización pertinente. Esta autorización acostumbra a ser una palabra de paso o palabra clave (contraseña) que el usuario especificará antes de acceder a la información y de la que el sistema operativo comprobará su veracidad.

CONTROL DE TÉCNICAS DE PROCESO AVANZADAS

Las grandes computadoras, las minicomputadoras y recientemente las computadoras personales están trabajando con técnicas de proceso avanzadas como la multiprogramación, el tiempo compartido, el *spooling*, el multiproceso y la memoria virtual.

Todas estas técnicas que son soportadas y controladas por el sistema operativo, hacen que las computadoras manifiesten mucha más potencia al operar y, sobre todo, permiten que se puedan ejecutar varios programas de forma concurrente.

La multiprogramación

Un sistema operativo con multiprogramación permite el almacenamiento en la memoria de una computadora de varios programas de usuario. La unidad central de proceso ejecuta un solo programa, pero cuando éste tenga que efectuar alguna operación en la que la C.P.U.

no intervenga y sí lo hagan los periféricos (por ejemplo, realizar una lectura o grabación en un disco o enviar datos a la impresora), ésta tiene tiempo de ejecutar otro programa que se encuentre en memoria hasta que el primer programa haya terminado la operación que efectuaba con el periférico, momento en que la C.P.U. dejará el segundo programa y reemprenderá la ejecución del primero en el punto donde lo dejó. La C.P.U. memorizará la dirección donde ha interrumpido la ejecución del segundo programa para que, cuando pueda volver a ejecutarlo, sepa dónde iniciar el proceso.

El tiempo compartido

El tiempo compartido es otra de las técnicas que permite un sistema operativo. Consiste en asignar un tiempo fijo de unidad central de proceso a cada uno de los programas que se van a ejecutar. Así, si hay que ejecutar diez programas, la unidad central de proceso ejecutará el primer programa durante dos segundos, por ejemplo (dos segundos de C.P.U. es mucho tiempo); una vez agotado el tiempo, interrumpirá la ejecución del primer programa e iniciará la ejecución del segundo durante dos segundos más, y así sucesivamente hasta llegar a la ejecución del décimo programa, momento en que, una vez finalizada su ejecución, volverá a empezar con la del primero.

La técnica de tiempo compartido varía sustancialmente de la de multiprogramación, ya que aquélla fija un tiempo de C.P.U. para cada programa, mientras que en ésta el tiempo puede ser variable según las entradas o salidas con periféricos que tengan los programas. Con las dos técnicas un sistema operativo consigue ejecutar varios trabajos simultáneamente y, como consecuencia, aumentar su potencia.

El spooling

La técnica del spooling surge ante la necesidad de agilizar la unidad central de proceso. La velocidad de impresión de los periféricos es baja, por lo que la C.P.U. tiene que esperar un cierto tiempo hasta que un programa imprima toda la información. Cuando son varios los programas que quieren imprimir, se crean conflictos de tiempo de espera y muchos otros si la impresión se quiere efectuar a través de la misma impresora. Frente a este problema, el sistema operativo permite establecer un área en disco donde se grabará toda la información dirigida a la impresora (el mismo sistema operativo se encarga de trasladar la información de memoria a disco). Esta área o zona de memoria se denomina *área de spooling* y en ella se grabarán los listados que la impresora tendrá que imprimir más tarde. De este modo, el tiempo de envío de la información a la impresora se reduce notablemente y no se producen tantos conflictos, porque disminuyen los tiempos de espera. Luego, cuando la computadora no esté tan cargada de trabajo, se podrán imprimir todos los listados por la impresora, siendo el mismo sistema operativo el encargado de leerlos en el área de spooling y enviarlos a la impresora.

El multiproceso

Esta técnica consiste en la ejecución de varios procesos al mismo tiempo, usando tantas unidades centrales de proceso como procesos tengamos que ejecutar.

La C.P.U. sólo soporta un proceso. Lo que ocurre en este caso es que hay varios procesadores o varias C.P.U. Al ser cada día más baratos los procesadores, esta técnica se utiliza cada vez más, ya que proporciona un rendimiento y una potencia mucho más amplios. Incluso puede variar la configuración física de las computadoras; en efecto, podemos encontrar una computadora con varios procesadores o varias computadoras conectadas entre sí con un solo procesador por cada una de ellas.

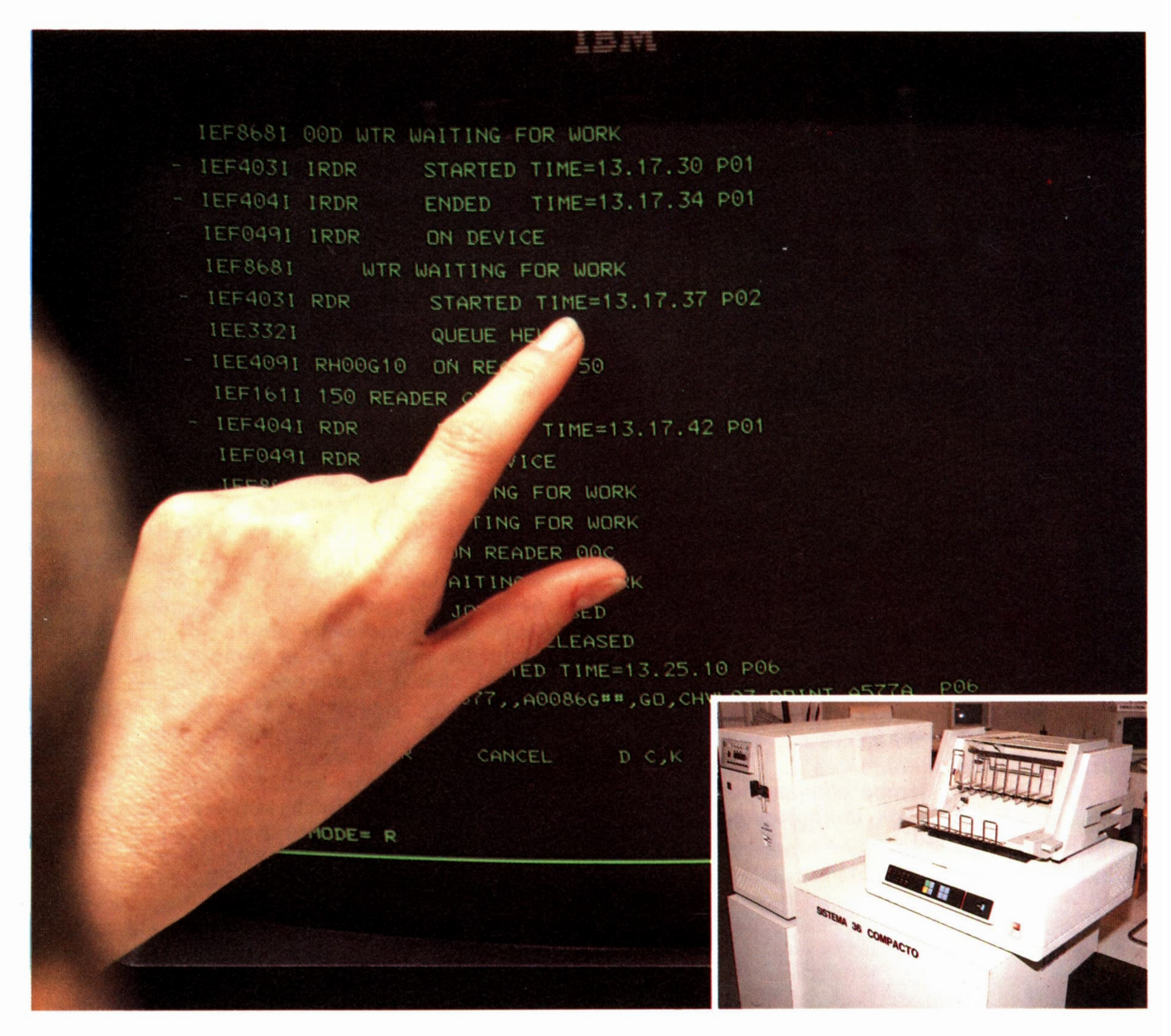
La diferencia fundamental con las demás técnicas es que con ésta se pueden procesar varios trabajos a la vez en un momento determinado mientras que con las otras sólo se puede procesar uno.

La memoria virtual

Con las técnicas que hemos visto hasta ahora, si un programa no cabe, debido a su extensión, en la parte de la memoria que tiene asignada, el programador no tiene otra solución que dividir el programa en módulos e ingeniár-

Un sistema operativo con multiprogramación

El sistema operativo es, en este caso, el que efectúa el control de todo el proceso y el que realiza la asignación de los recursos para los diferentes procesos.
Se puede hacer la comparación de este sistema de trabajo con el que se lleva a cabo en un salón de peluquería. Cada una de las peluqueras sería la unidad central de proceso. Atienden a las clientes y, cuando una de éstas está en el secador, por ejemplo, tienen tiempo de atender a otra y así sucesivamente. La encargada de la peluquería sería el sistema operativo y decidiría el orden por el que se atendería a las clientes, repondría material cuando fuera necesario, etcétera.
Esta técnica no se ha inventado con las computadoras, sino que ha sido una adaptación a ellas para aprovechar al máximo su potencia.



En la pantalla de esta computadora, utilizada en régimen de tiempo compartido, aparece especificado el tiempo que cada usuario, identificado por un número, se ha servido de la máquina.

selas para que, cuando acabe la ejecución del primer módulo, se cargue en el mismo espacio de memoria el segundo módulo y así sucesivamente hasta que termine la ejecución del programa. Este proceso de descomposición es muy complejo y sólo podrían hacerlo personas que conocieran a fondo el sistema. Para paliar este inconveniente se incorporó la técnica de la memoria virtual en los sistemas operativos. Esta técnica realiza el trabajo que hacía el programador antes de incorporar la memoria virtual al sistema operativo de la computadora.

Supongamos un programa que ocupe 80 Kbytes. Con la técnica de la memoria virtual, el programa se dividirá en páginas (cada página de 8 Kbytes, por ejemplo), de modo que podamos tener en memoria las tres primeras páginas del programa, es decir, 24 Kbytes. Al finalizar la ejecución de estos módulos del programa, se leen automáticamente tres páginas más del programa de la memoria virtual (generalmente el disco donde está almacenado el programa) y se cargan en lugar de las tres ejecutadas y actualizadas con anterioridad en la memoria virtual. Así se van sucediendo estos procesos sin la intervención del programador.

Hay incluso sistemas operativos que controlan las páginas que permanecen más tiempo en memoria, para dejarlas ya fijas en la memoria central en un proceso posterior.

Con esta técnica, el espacio en disco se comporta como si fuera la memoria central, pero con la salvedad de que no se accede directamente a la información que el disco contiene, sino que la información requerida se lee y se traslada a la parte correspondiente de la memoria central y la información que había en esta parte se salva en el disco. Por eso no puede llamársele memoria central y se le da el nombre de memoria virtual.

Conceptos básicos en la programación de computadoras

Ninguna computadora por sí sola puede realizar las tareas que se le encomiendan; para que la máquina pueda trabajar es necesario que alguien le indique qué debe hacer y cómo tiene que hacerlo. El que da las instrucciones a la computadora es el programador, persona versada en las técnicas mediante las cuales se pueden dar órdenes a la máquina. Al conjunto de estas técnicas se le llama *programación*.

Los elementos fundamentales de la programación son los programas. Un programa es un conjunto de instrucciones o sentencias perfectamente comprensibles por la computadora que sirve para que ésta pueda realizar un determinado trabajo.

Si queremos que una computadora lleve a cabo un proceso, antes que nada habrá que programarla para ello. Tendremos que construir un programa que le indique qué pasos ha de seguir para efectuar dicho proceso de un modo totalmente correcto.

Construcción del programa

La construcción de un programa no es una tarea trivial, sino que requiere un análisis y un conocimiento claro y profundo del problema que se quiere resolver, así como el dominio de un lenguaje de programación que permita transmitir las instrucciones a la computadora.

El primer paso necesario para diseñar un programa es el planteamiento y el análisis del problema que se quiere resolver. Una vez clarificada la cuestión, se construirá un procedimiento de cálculo, paso a paso, mediante el cual se pueda obtener la solución del problema. Puede suceder que el problema que queremos resolver no tenga solución; en este caso no será necesaria la construcción del programa. Después de construir el procedimiento, transformaremos todos los pasos que lo componen en instrucciones o sentencias propias de un lenguaje de programación que pueda ser comprensible para la computadora para la que se construye el programa.

Hay técnicos en programación de computadoras que, por su larga experiencia o por la fuerza de la costumbre, no realizan el primer paso de análisis y construcción de un procedimiento para resolver el problema, sino que proceden directamente a la construcción del programa con instrucciones o sentencias del lenguaje de programación. No es aconsejable que las personas que se inician en el aprendizaje de la programación de computadoras o aquellos que tienen pocos años de experiencia en este campo prescindan de este primer paso, ya que esta omisión dificulta la programación propiamente dicha, excluye la documentación del programa y aumenta los errores tanto en compilación, si la hay, como en la ejecución del programa.

Tan importante para la construcción de programas es el conocimiento de las técnicas de programación como de los conceptos básicos sin los cuales la creación de los programas sería muy difícil o casi imposible. El conocimiento de estos conceptos ayudará al futuro programador, o al programador en general, a comprender de qué forma trabaja la computadora.

En este capítulo se describen los conceptos básicos fundamentales que hay que conocer antes de entrar en el aprendizaje de un lenguaje de programación que nos permita construir programas.

ALGORITMOS Y DIAGRAMAS DE FLUJO

Los algoritmos son los procedimientos que se construyen para la resolución de cualquier problema. Cuando en el capítulo anterior hablábamos del primer paso que hay que realizar en la construcción de un programa, nos estábamos refiriendo a la creación de un algoritmo.

El algoritmo no es un concepto proveniente del campo de la computación, sino que es un término matemático.

En computación los algoritmos generalmente se representan en forma gráfica y entonces reciben el nombre de *diagramas de flujo*.

Los diagramas de flujo constituyen el mejor método que se puede utilizar en la construcción de programas y, como toda metodología, dispone de unos convenios y de una simbología específicos para su desarrollo.

Simbología de un diagrama de flujo

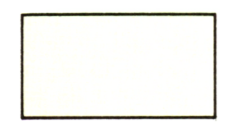




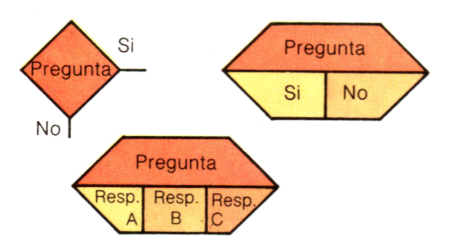
Con esta señal se quiere indicar el principio o el fin del diagrama de flujo.



Este símbolo se utiliza para indicar entrada de datos a la computadora, ya sea desde el teclado, ya sea desde otra computadora. Esta función se puede indicar mediante otros símbolos (dibujo de un terminal de computadora o de un teclado), pero el que muestra la figura es el más usado.

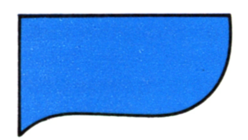


Este rectángulo nos informa de las órdenes ejecutivas, tales como asignaciones de datos a una variable, cálculos con variables u operaciones con datos.



Las computadoras actuales ofrecen enormes posibilidades para la generación de gráficos. En general, se utilizan elementos básicos muy sencillos que se organizan bajo las órdenes del programa. En la serie de dibujos superior puede verse una elaboración gráfica de las sucesivas fases del desarrollo de una mariposa realizada con una computadora. A la izquierda, resultado de un programa que genera una serie de cuadrados elementales con «efecto de torsión».

Con estas figuras señalamos cuestiones o preguntas. No son sino la representación de puntos dentro del diagrama de flujo, en los que se pueden tomar varias decisiones según la situación en la que el usuario se encuentre.

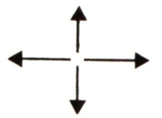


Este símbolo se usa para mostrar salidas impresas, porque representa un listado efectuado por la computadora.





Ese signo se utilizará como conector entre varias posiciones de un diagrama de flujo. Puede ocurrir que, al diseñar un diagrama, éste sea tan extenso que no quepa en la hoja de papel. Terminaremos la porción de diagrama que nos queda con unos de estos símbolos a los que daremos un número, 1 por ejemplo, y continuaremos la representación del diagrama de flujo en otro papel empezando con el mismo símbolo, 1, con que hemos finalizado la porción anterior. Estos símbolos indican que el diagrama continúa en otro espacio.



Estos signos son fundamentales, e indican la dirección de los pasos que hay que seguir en cada caso.

Los diagramas de flujo no sólo se utilizan en computación, sino que se emplean siempre que se quiera solucionar un problema, aunque para ello no se requiera el uso de una computadora.

Construcción de un diagrama de flujo

Ilustraremos este apartado con un ejemplo. Construiremos el diagrama de un proceso muy sencillo y que diariamente efectúan millones de personas: cruzar una calle. El problema es cruzar la calle, y ahora se describirán los pasos necesarios para solucionar este problema, es decir, cruzar la calle sin peligro de ser atropellados por un automóvil.

Como vemos en la **figura 1** *(pág. 180)*, el primer paso será preguntarnos si hay algún semáforo en la calle. Hay dos posibles respuestas a esta pregunta; según sea afirmativa o negativa, el diagrama especificará los pasos que habrá que realizar. Si no hay semáforo, miraremos a la izquierda para comprobar que no se acerca ningún vehículo. Si viene alguno, tendremos que esperar hasta que no se acerque ninguno. En este momento efectuaremos la misma ope-

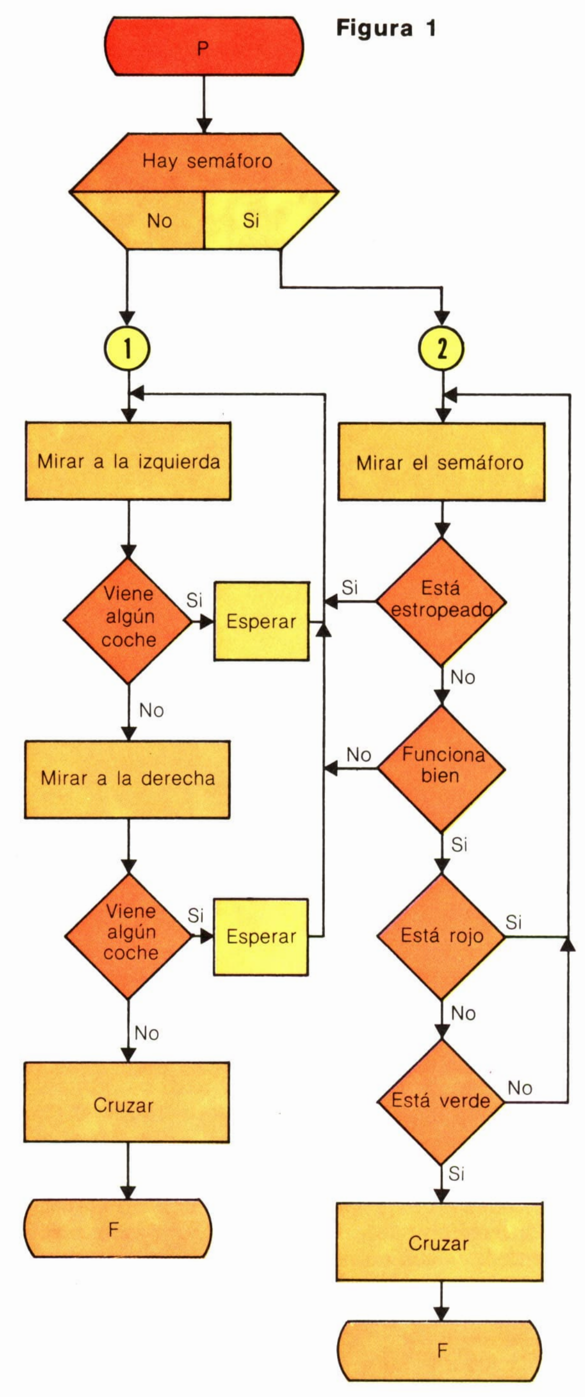


Figura 1

ración en la derecha: si viene algún vehículo, esperaremos y volveremos a efectuar la misma operación en la izquierda, porque en este tiempo de espera puede haber variado la situación de este lado. Iremos efectuando alternativamente estas operaciones hasta que llegue el momento en que no veamos ningún vehículo ni por el lado derecho ni por el izquierdo, momento en el que cruzaremos y con ello se habrá terminado el proceso. En esta parte del proceso del diagrama de flujo, encontramos dos órdenes ejecutivas: mirar a la izquierda y mirar a la derecha. No importa el orden de colocación de estas dos órdenes, ya que su posición no modifica el resultado final. El hecho de que en este caso se haya colocado primero la orden de mirar a la izquierda responde a un criterio práctico: generalmente, es de ese lado de donde vienen los vehículos que están más cerca del individuo que pretende cruzar.

En el caso de que exista semáforo, nos fijaremos si está funcionando o está averiado; en este último caso procederíamos como si no lo hubiere. (La flecha que sale de la respuesta «sí está estropeado» y va al principio de la parte del diagrama que trata el caso de que no haya semáforo señala este salto.) Si no está estropeado, nos fijaremos en su funcionamiento. Si no funciona correctamente, procederemos como en el caso anterior. Si funciona correctamente, nos preguntaremos si está rojo; si la respuesta es afirmativa, repetiremos esta operación hasta que la respuesta sea negativa. En tal caso, investigaremos si está verde. Si no estuviera verde, sólo puede estar ámbar. Llegados a este extremo, volveremos otra vez al punto inicial. Si el semáforo está verde, cruzaremos y finalizará el proceso.

Este diagrama de flujo muestra un proceso de la vida real, no susceptible, por tanto, de ser realizado por una computadora que únicamente ejecuta o interpreta procesos o instrucciones numéricos. No es secuencial, sino que presenta algún retroceso en su estructura. Estos retrocesos evitan las repeticiones. El término utilizado para indicar los saltos hacia adelante o hacia atrás de un diagrama de flujo es *bifurcación*. En el diagrama que hemos construido a modo de ejemplo, cuando vienen vehículos por la derecha, se da una bifurcación.

Diagramas de flujo de algoritmos no numéricos

Los diagramas de algoritmos no numéricos son ininterpretables por la computadora. Los ejemplos del apartado anterior responden a diagramas de algoritmos no numéricos. Podemos mostrar otro ejemplo: el proceso que hay que seguir para coser un botón de dos agujeros en una prenda de vestir (véase **figura 2**).

Este algoritmo difícilmente puede ser interpretado por la computadora. No existe ningún lenguaje de programación que permita la construcción de un programa a partir de este diagrama de flujo.

Se han descrito los pasos que se efectúan generalmente para coser un botón, pero podríamos haber especificado todavía más. Por ejemplo, al final del diagrama encontramos la orden «rematar»; se puede sustituir esa orden por to-

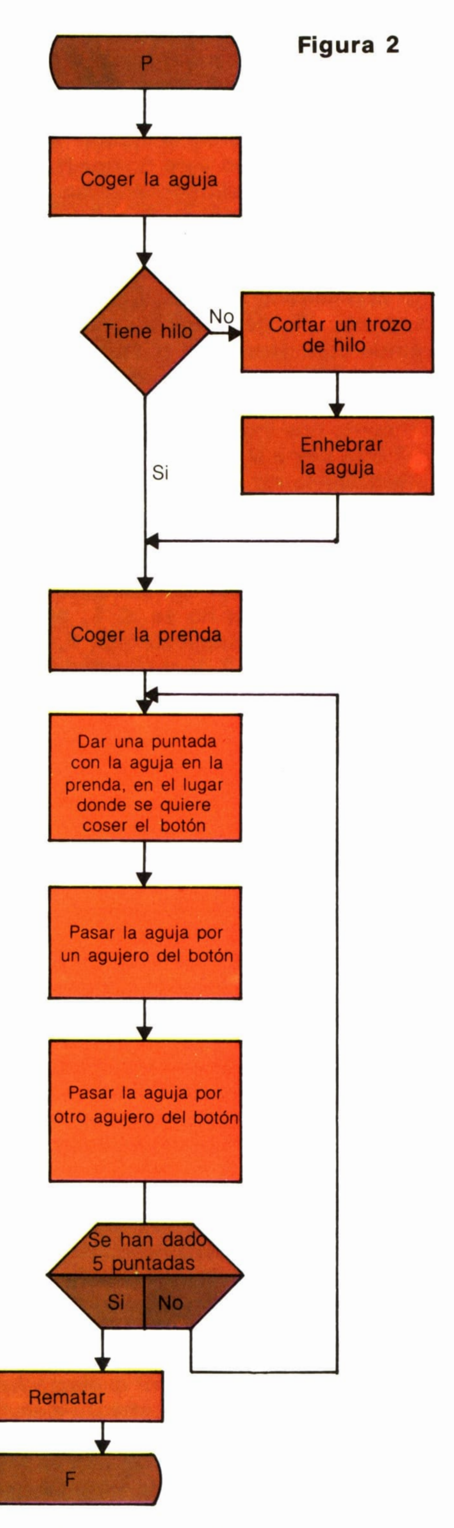


Figura 2

das las acciones necesarias para su realización. Incluso podríamos descomponer mucho más los pasos que se han ido disponiendo en el diagrama en otros más elementales.

En los algoritmos no numéricos, es muy difícil identificar cada uno de los pasos elementales de una forma objetiva; en cambio, en los algoritmos numéricos esto no es así, como comprobaremos en el capítulo siguiente.

En este diagrama de flujo, al igual que en el de «cruzar la calle», se presenta una construcción muy común en todos los diagramas, llamada *bucle*. El bucle es la repetición de un número de pasos del diagrama hasta que se cumpla una determinada condición que cierre el ciclo repetitivo. En este caso el bucle está formado por los cuatro pasos marcados con un asterisco; el final del bucle es la pregunta de control del número de veces que se han realizado uno o varios pasos. Mientras no se hayan efectuado cinco puntadas con la aguja en la prenda, no se saldrá del bucle y se irán repitiendo los pasos marcados con el asterisco. Si no se utilizara este sistema tendríamos que repetir secuencialmente todos los pasos, lo que equivaldría a la representación de quince pasos, cuando, con el bucle, con cuatro tenemos suficiente. En este mismo capítulo se describirá con más detalle el procedimiento del bucle.

Diagramas de flujo de algoritmos numéricos

Los algoritmos numéricos siempre tienen correspondencia con problemas matemáticos, por lo que sus diagramas de flujo sí pueden representarse a través de una computadora.

Las matemáticas aparecen en todas las actividades que el hombre efectúa. Los diagramas de algoritmos numéricos aparecerán también en todas estas actividades y la computadora intervendrá en todos los campos que afectan al hombre.

(A partir de este capítulo nos referiremos a los diagramas de flujo de algoritmos numéricos simplemente con el término diagrama.)

Así como la mayoría de problemas matemáticos presentan una sola solución, pero podemos llegar a ella a través de varios procedimientos, también podremos construir varios diagramas de un mismo problema. De ahí que existan distintos programas para resolver un único problema según los programadores que los hayan diseñado.

Propondremos como ejemplo de diagrama un problema sencillo: el cálculo del factorial de cualquier número.

En primer lugar, explicaremos qué es el factorial de un número. Para calcular el factorial de un número, por ejemplo el 6, efectuaremos el siguiente producto: $6 \times 5 \times 4 \times 3 \times 2 \times 1$. El diagrama de resolución de este caso podría ser el de la **figura 3**.

Este diagrama sirve solamente para realizar el cálculo del factorial de 6 y no lo podemos usar para el cálculo del factorial de cualquier otro número. Cuando se construye un diagrama de resolución de un problema numérico se pro-

cura siempre que contemple la mayoría de casos. Así, nunca se construirá un diagrama para resolver un solo caso, como ocurre en el ejemplo que hemos propuesto, sino que se compondrá un diagrama que contemple la resolución de cualquier factorial de un número. Esto no quiere decir que el que hemos construido no sea correcto; lo que ocurre es que no es práctico.

Sin embargo, con pocos cambios, este diagrama podría convertirse en genérico.

Para ello, fijémonos en una de las características fundamentales sin la cual no se podría construir ningún diagrama de algoritmo numérico. Esta característica es el uso de variables numéricas.

Una variable numérica se representa siempre con una letra o con un nombre, con los cuales se significa un valor que no tiene por qué ser constante, sino que puede variar. Por ejemplo, en el caso que estamos estudiando usamos la variable F, que almacena el resultado del factorial, y la variable N, en la que almacenamos el número del cual queremos obtener el factorial. Si analizamos dicho diagrama, veremos que al ejecutar el paso 1, mover 6 a N, N pasará a valer 6, y al ejecutar el paso 2, mover 6 a F, F será igual a 6; al ejecutar el paso 3, obtendremos

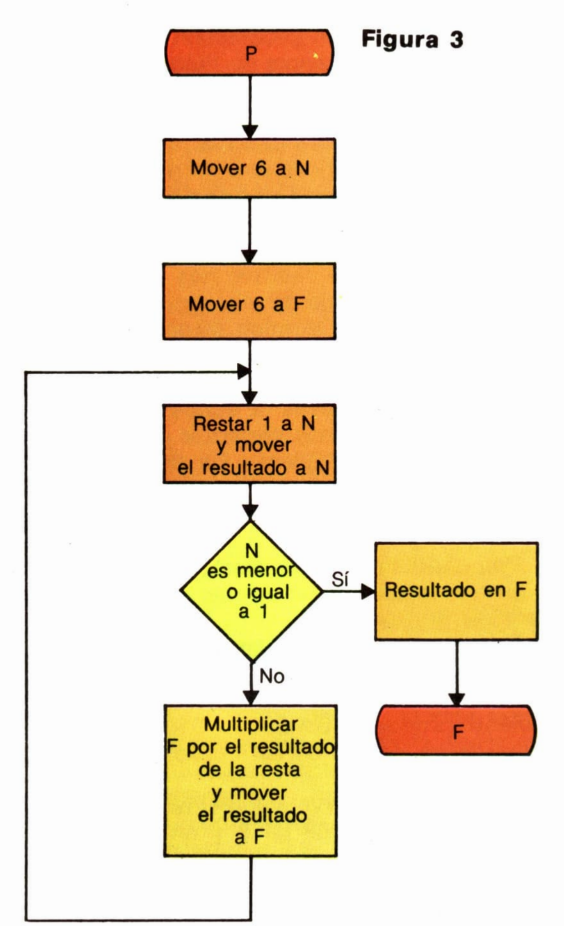


Figura 3

como resultado 5, porque N valía 6, con lo cual N cambiará su valor a 5; en el paso 4 comprobamos que el resultado de la resta no es ⩽1. Seguimos con el paso 5: al ejecutarlo, deberemos multiplicar F, que vale 6, por N, que vale 5, y el resultado lo almacenaremos en F, con lo cual su valor pasará de 6 a 30. Siguiendo con el diagrama volveremos a ejecutar el paso 3 y obtendremos como resultado 4, porque N valía 5. El resultado de la resta de N valdrá 4 y, como no es ⩽1, debemos seguir, con lo cual tendremos que ejecutar el paso 5 y multiplicar F, que vale 30, por N, que vale 4, y el resultado, 120, almacenarlo en F. Volviendo al paso 3, tal y como nos indica el diagrama deberemos restar 1 otra vez a N, con lo cual pasará a valer 3; como todavía no es ⩽ a 1, seguiremos con el paso 5 y multiplicaremos F, que vale 120, por N, que vale 3, y el resultado, 360, almacenarlo en F. Volviendo otra vez al paso 3, deberemos restar 1 a N, con lo cual su valor ahora será 2. Como todavía no es ⩽ 1, seguiremos con el paso 5 y multiplicaremos F, que vale 360, por N, que vale 2, y el resultado, 720, almacenarlo en F. Volviendo otra vez al paso 3 deberemos restar 1 a N, con lo cual su valor pasará a ser 1. En el paso 4 investigamos si N ⩽ 1; al cumplirse esta condición, ejecutaremos el paso 6, que nos dice que el resultado que estamos buscando se encuentra en F. Por lo tanto, el resultado será 720, porque F=720.

Observamos que las variables N y F van cambiando su valor a medida que se van ejecutando todos los pasos del diagrama.

Veamos ahora cómo sería el diagrama para calcular el factorial de cualquier número. En su construcción consideraremos que posteriormente se va a convertir en un programa y, por lo tanto, será ejecutado por una computadora.

Analizando el diagrama de la **figura 4**, podemos ver que en el primer paso se efectúa una lectura de N; esto significa que se va a leer un número, precisamente del que se quiere obtener el factorial, y que será asignado a la variable N. Se puede entrar este número a la computadora, manualmente desde teclado o a través de un fichero donde estará almacenado, con lo que se le da un carácter genérico al diagrama, ya que N, cada vez que ejecutemos los pasos del diagrama en el paso 1, podrá tener cualquier valor y por lo tanto dicho diagrama y el posterior programa que podremos desarrollar a partir de él nos servirán para calcular el factorial de cualquier número.

En el paso 2 se comprueba si el número almacenado en la variable N es un número natural, es decir, si es un número positivo y sin decimales, incluido el cero. Si no lo es, se deberá leer otro número, porque el factorial sólo lo calcularemos de números naturales. Si se cumple la condición, el siguiente paso es el número 3, en el que se almacena en F el contenido de N;

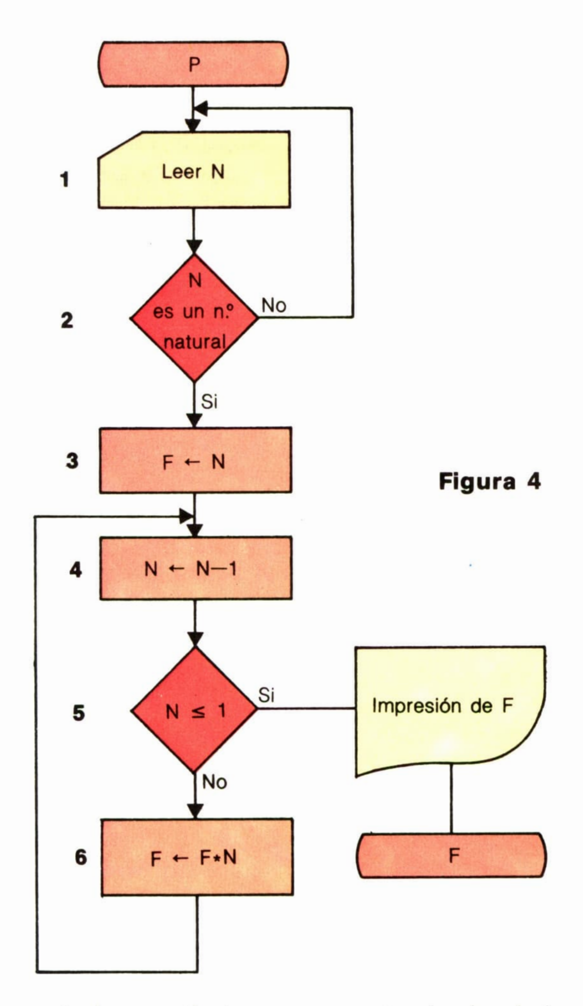


Figura 4

es decir, a partir de ese momento el valor de la variable F será igual al de la variable N.

Al ejecutar el paso 4 lo que se hace es disminuir el valor de N en 1. La terminología empleada en este paso es la terminología que se usa en todos los lenguajes de programación. Se asigna a una variable el valor de ella misma, pero restándole la unidad. Por ejemplo, si antes de ejecutar este paso N valía 5, al ejecutarlo se resta 1 a 5 y el resultado, 4, se asigna a N, con lo cual pasará a ser 4. Este paso es el mismo que el paso 3 del diagrama anterior. Como vemos se utiliza el signo ←, cuyo significado es «sustituir por», y no el de «igual a»; aunque después, cuando se construya el programa en el lenguaje de programación, este paso se sustituye por el signo «=».

El siguiente paso es el 5, en el que se comprueba si el valor de la variable N es menor o igual a 1; esta condición se representa por el signo ⩽. Si se cumple la condición, quiere decir que ya se ha terminado el cálculo y por lo tanto se ejecutará el paso 7, con lo cual se imprimirá el resultado que tendremos en F. Si no se cumple la condición, se ejecutará el paso 6, en el que se realizará la multiplicación del valor de la variable F por el de la variable N y el resultado se almacenará en la variable F. (Como se puede comprobar, el signo de multiplicar se representa por un asterico; esto es debido a que, si se utiliza el carácter ×, éste podría confundirse con la letra X. Esta convención se utiliza en todos los lenguajes de programación, de ahí que también la utilicemos en los diagramas.)

Seguidamente se volverá a ejecutar el paso 3 y así sucesivamente.

Si comparamos el diagrama anterior con éste, veremos que las únicas diferencias se encuentran al principio, ya que el primer paso del primer diagrama se desglosa en dos en el segundo, precisamente los pasos que sirven para hacer que el diagrama sea genérico.

Construcción óptima de un diagrama de flujo

La única manera de poder saber si un diagrama de flujo está bien hecho o no es seguirlo paso a paso e ir comprobando si contempla todas las posibilidades que podemos encontrarnos dentro de la resolución de un problema. Por ejemplo, en el diagrama del cálculo del factorial de un número, si no especificásemos el segundo paso, ello querría decir que se nos podría presentar el caso de tener que calcular el factorial de un número con decimales y, por lo tanto, el diagrama construido no nos sería útil. Sin embargo, al especificar dicho paso, contemplamos el hecho de que se nos presente tal caso y le damos una solución: volver a leer otro número.

Como ya hemos dicho, en el desarrollo de un diagrama siempre intervendrán variables numéricas y en ellas normalmente se obtendrá el resultado. Por lo tanto, siempre se tendrá que controlar el valor de tales variables (y seguirlo).

Por ejemplo, para seguir paso a paso el diagrama del cálculo del factorial de cualquier número y, por lo tanto, comprobar si está bien hecho, se darán los pasos que se indican a continuación.

Leeremos el número y lo almacenaremos en N. Supongamos que hemos leído el número 7; entonces

N = ~~7~~ ~~6~~ ~~5~~ ~~4~~ ~~3~~ ~~2~~ 1
F = ~~7~~ ~~42~~ ~~210~~ ~~840~~ ~~2520~~ 5040

El resultado del problema, siguiendo el diagrama, será 5040.

Suponiendo que el número leído sea 5, entonces:

N = ~~5~~ ~~4~~ ~~3~~ ~~2~~ 1
F = ~~5~~ ~~20~~ ~~60~~ 120

el resultado será 120.

He aquí dos ejemplos cuyos resultados deberemos comparar con los resultados de estos mismos ejemplos, pero calculados por el método normal. Es decir, en el primer ejemplo tene-

mos que calcular el factorial de 7, por lo tanto deberemos hacer $7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$, con lo que comprobamos que hemos obtenido el mismo resultado. En el segundo ejemplo deberemos calcular el factorial de 5: $5 \times 4 \times 3 \times 2 \times 1 = 120$, con lo que también comprobamos que se ha obtenido el mismo resultado. Calculando un par de ejemplos más, comprobaremos que también dan el mismo resultado en cada caso, con lo que ya podemos decir que el diagrama está bien construido.

Para aclarar el seguimiento de un diagrama de flujo explicaremos paso a paso uno de los dos casos antes descritos; por ejemplo, en el caso de 5! (factorial de 5).

Antes que nada, se analiza el diagrama y se escriben todas las variables que intervienen. En este caso sería:

N =   F =

A continuación viene el seguimiento del diagrama:

• Ejecutar el paso 1, con lo que se leerá un número y se almacenará en N. En este caso será:

N = 5   F =

• Ejecutar paso 2; el número es natural, por lo tanto seguimos.

• Ejecutar paso 3, con lo que igualamos F a N. En este caso:

N = 5   F = 5

• Ejecutar paso 4, con lo que restamos 1 a N. En este caso:

N = ~~5~~ 4   F = 5

N valdrá 4 y no 5, por eso tachamos el 5 y ponemos 4.

• Ejecutar paso 5. Como N no es $\leqslant 1$, seguimos con el paso 6.

• Ejecutar paso 6. Multiplicaremos F por N y el resultado lo almacenaremos en F:

N = ~~5~~ 4   F = ~~5~~ 20

F pasará a valer 20, resultado de multiplicar 5 por 4, y por lo tanto tacharemos el 5 y pondremos 20.

• Ejecutar otra vez el paso 4, porque el diagrama nos lleva ahí. Por lo tanto, volveremos a restar 1 a N. En este caso:

N = ~~5~~ ~~4~~ 3   F = ~~5~~ 20

N pasará a valer 3, por eso tacharemos el 4 y pondremos 3.

• Ejecutar otra vez el paso 5. Como N no es $\leqslant 1$, seguiremos con el paso 6.

• Ejecutar otra vez el paso 6. Multiplicaremos F, que vale 20, por N, que vale 3, y el resultado lo almacenaremos en F. En este caso:

N = ~~5~~ ~~4~~ 3   F = ~~5~~ ~~20~~ 60

F pasará a valer 60, resultado de multiplicar 20×3, y por lo tanto tacharemos 20 y pondremos 60.

• Ejecutar nuevamente el paso 4. Volveremos a restar 1 a N. En este caso:

N = ~~5~~ ~~4~~ ~~3~~ 2   F = ~~5~~ ~~20~~ 60

N pasará a valer 2, por eso tacharemos el 3 y pondremos 2.

• Ejecutar otra vez el paso 5. Como N no es $\leqslant 1$, seguiremos con el paso 6.

• Ejecutar de nuevo el paso 6. Multiplicaremos F, que vale 60, por N, que vale 2, y el resultado lo almacenaremos en F. En este caso:

N = ~~5~~ ~~4~~ ~~3~~ 2   F = ~~5~~ ~~20~~ ~~60~~ 120

F pasará a valer 120, resultado de multiplicar 60×2, y por lo tanto tacharemos 60 y pondremos 120.

• Ejecutar otra vez el paso 4. Volveremos a restar 1 a N. En este caso:

N = ~~5~~ ~~4~~ ~~3~~ ~~2~~ 1
F = ~~5~~ ~~20~~ ~~60~~ 120

N pasará a valer 1, por eso tacharemos el 2 y pondremos 1.

• Ejecutar de nuevo el paso 5. Como N es $\leqslant 1$, seguiremos con el paso 7.

• Ejecutar paso 7. Se imprimirá F, cuyo valor será 120, y éste será el resultado y el último paso ejecutado.

Fijémonos en los dos últimos valores de N y F: N = 1 y F = 120.

El valor de N es fundamental en el diagrama, porque en el paso 5, según N sea $\leqslant 1$, querrá decir que ya hemos calculado el resultado o no, con lo cual el valor de F es ya el resultado calculado.

Elementos principales de un diagrama de flujo

Vamos a ver a continuación las partes o elementos de todo diagrama de flujo que comportan una mayor dificultad en el momento de su construcción. Estos elementos son los siguientes:

• Los bloques de decisión:
  — sencillos;
  — compuestos;
  — múltiples.
• Las expresiones aritméticas.
• Variables con índice.
• Los bucles:
  — sencillos;
  — anidados.

A continuación analizaremos cada uno de ellos.

LOS BLOQUES DE DECISIÓN

Los bloques de decisión son elementos de un diagrama de flujo en los que se produce la toma de una determinada decisión: si se cumple o no una o unas determinadas condiciones.

Condiciones simples

Las condiciones simples son elementos de un diagrama en los cuales solamente pueden tomarse dos caminos o dos decisiones, que a

su vez dependen de que se cumpla sólo una condición.

En estas condiciones, siempre se utilizan los operadores aritméticos de relación, que son los siguientes:

$<$	menor que
$>$	mayor que
$=$	igual a
$\leqslant$	menor o igual que
$\geqslant$	mayor o igual que
$\neq$	diferente o no igual a

Generalmente, la condición se construye mediante tres elementos:

1	2	3
expresión aritmética	operador de relación	expresión aritmética

Veamos un ejemplo de bloque de decisión por cada uno de estos operadores de relación:

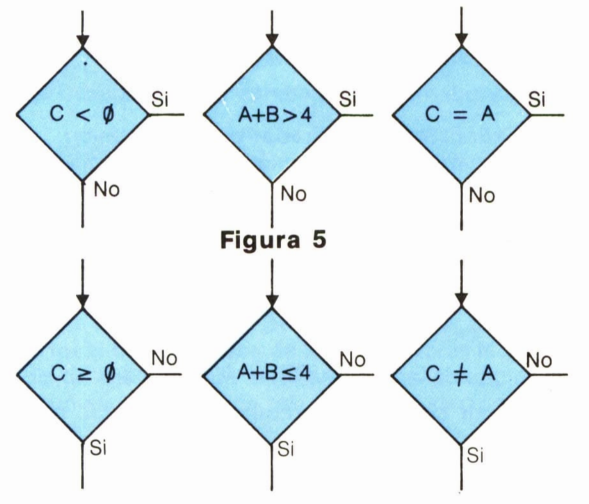


Figura 5

En principio fijémonos en que todos los bloques tienen solamente dos alternativas, sí o no, y en que todos están en el mismo lugar. Esto no quiere decir que si las alternativas estuvieran en lugares diferentes el bloque no sería el mismo. Por ejemplo, supongamos la condición C = A, cuyo bloque de condición es el de la **figura 6**, este bloque sería el mismo que todos los de la **figura 7**, sólo que cambiaría la representación de las decisiones que hay que tomar, pero éstas serían las mismas. Por lo tanto, entre la **figura 6** y la **7** tenemos seis maneras diferentes de representar un mismo bloque de decisión. Normalmente se acostumbra adoptar uno de estos seis y así siempre se hace el mismo. Por eso los bloques de la **figura 5** están todos dibujados de la misma manera.

En cuanto a los operadores de relación, fijémonos que son opuestos entre sí:

el signo de $=$ es opuesto al de $\neq$
el signo de $<$ es opuesto al de $\geqslant$
el signo de $>$ es opuesto al de $\leqslant$.
Es decir:
$5 > 0$ y por lo tanto no puede ser $\leqslant 0$
$5 < 6$ y por lo tanto no puede ser $\geqslant 6$
$5 = 5$ y por lo tanto no puede ser # 5

Por lo tanto, si prestamos atención a los bloques de decisión de la **figura 5**, también los podríamos construir como en la **figura 8**:

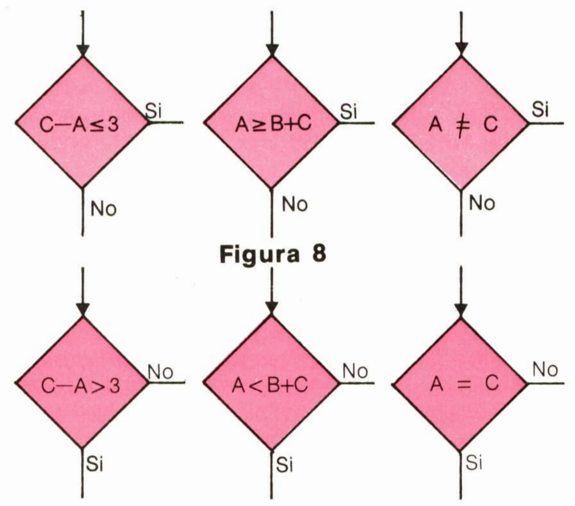


Figura 8

Observemos que en cada uno de estos bloques hemos variado la condición; la hemos cambiado por su opuesta, lo cual nos ha obligado a cambiar los caminos. Por ejemplo, en el primer caso, con C > 0, si se cumple, se ejecu-

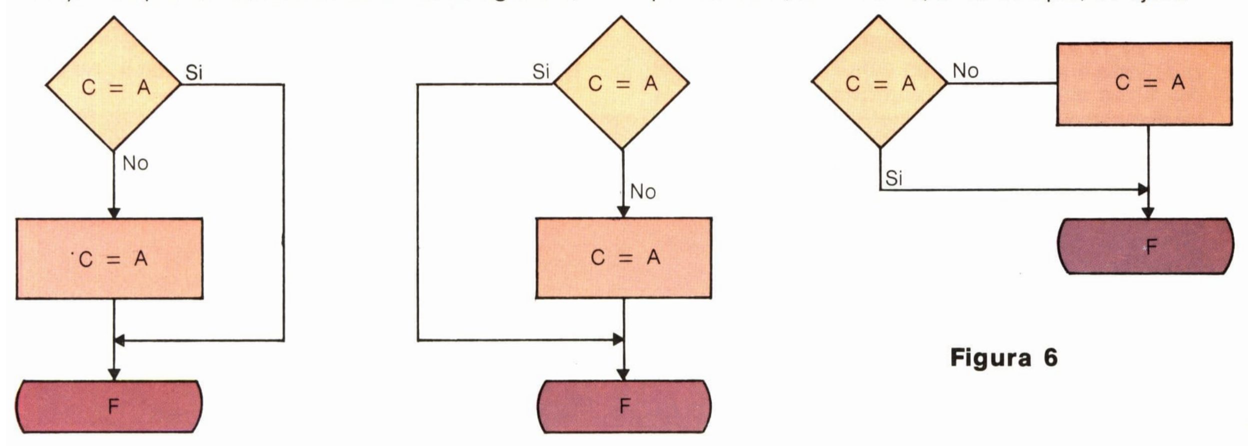
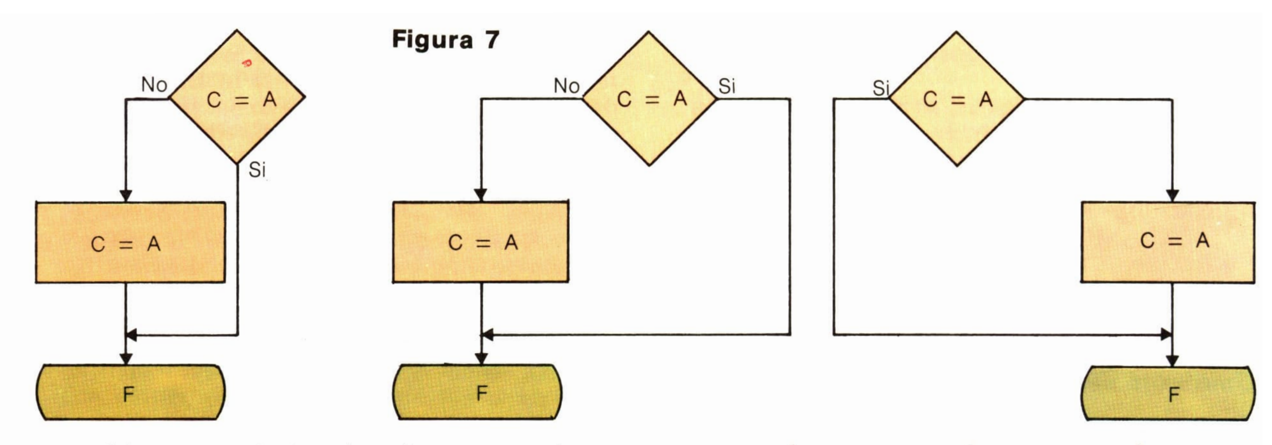


Figura 6



tará la rama de la derecha; si no se cumple, se ejecutará la rama de abajo. En cambio, con la condición C ⩾ 0, si se cumple, se ejecutará la rama de abajo y, si no se cumple, se ejecutará la rama de la derecha. Sin embargo, no hay que tocar nada de lo representado en las ramas, si no solamente del bloque de decisión.

Por ejemplo, supongamos el diagrama de la **figura 9A** *(pág. 188)*, en el que nos encontramos con un bloque de decisión de la condición A > 10.

Este mismo diagrama se representa en la **figura 10A** *(pág. 188)*, con la sola diferencia del bloque de decisión, ya que la condición es la opuesta, A ⩽ 10, y por lo tanto varían las salidas «sí» y «no», pero no varían los pasos que hay que seguir tanto a la derecha como hacia abajo.

Condiciones compuestas

Las condiciones compuestas son elementos de un diagrama de flujo donde solamente pueden tomarse dos caminos o dos decisiones, según se cumplan dos o más condiciones. Es decir, están formadas por la unión lógica de dos o más condiciones sencillas. Dicha unión se efectúa mediante los tres operadores lógicos fundamentales, que son:

OR en lenguaje corriente o
AND en lenguaje corriente y
NOT en lenguaje corriente no

Por lo tanto, la construcción se efectuará mediante tres elementos:

1	2	3
condición simple	operador lógico	condición simple

Así como la condición simple, según hemos visto antes, se subdividía en:

Expresión aritmética	Operador de relación	Expresión aritmética

ahora podemos subdividirlas todavía más:

Expresión aritmética	Operador de relación	Expresión aritmética
	Operador lógico	
Expresión aritmética	Operador de relación	Expresión aritmética

Por ejemplo:

A + C > B   OR   A + C < B

sería una condición compuesta expresada con el operador lógico OR. Esta misma condición normalmente se expresa como A + C > B o A + C < B. Esto significa que se cumplirá la condición compuesta si se cumple la primera condición sencilla o si también se cumple la segunda condición sencilla.

Para entrar con más profundidad en este tema, estudiaremos estos tres operadores lógicos.

Sirven para enlazar condiciones simples, que pueden cumplirse o no, es decir, que pueden ser verdaderas o falsas. Por ejemplo, si nos encontramos con la condición C > D y si C = 5 y D = 6, tal condición será falsa, es decir, no se

Tabla de resultados de los operadores lógicos

A	***B***	***NOT A NO A***	***NOT B NO B***	***A OR B A o B***	***A AND B A y B***
V	F	F	V	V	F
F	V	V	F	V	F
F	F	V	V	F	F
V	V	F	F	V	V

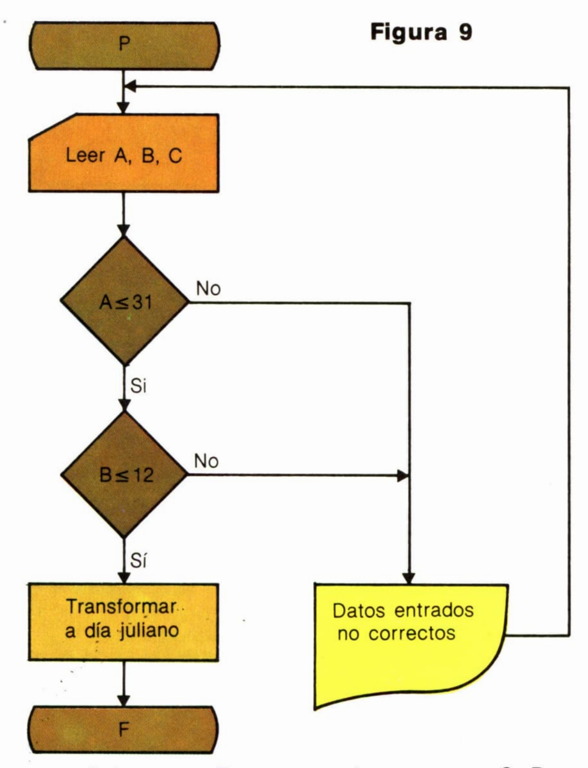


Figura 9

cumplirá porque 5 nunca será mayor que 6. Por lo tanto, estos operadores lógicos, al unir condiciones con resultado diferente, pueden también dar resultados diferentes. Para ver todos los posibles resultados que pueden dar, construiremos la **tabla de resultados de los operadores lógicos**, de tal manera que tendremos dos condiciones simples A y B, cuyos resultados pueden ser verdaderos (representados mediante una V) o falsos (representados mediante una F). El número de combinaciones que pueden resultar al juntarlas son 4, tal como se comprueba en dicha tabla. La primera combinación es la condición A verdadera y B falsa; entonces el resultado de NO A será falso, y el de NO B verdadero, es decir, justamente al contrario; por lo tanto, en todas las demás combinaciones siempre será el contrario. El resultado de A o B siempre será verdadero cuando una de las dos o las dos condiciones sean verdaderas y será falso cuando las dos sean falsas. El resultado de A y B siempre será falso cuando alguna de las condiciones sea falsa y será verdadero cuando las dos condiciones sean verdaderas.

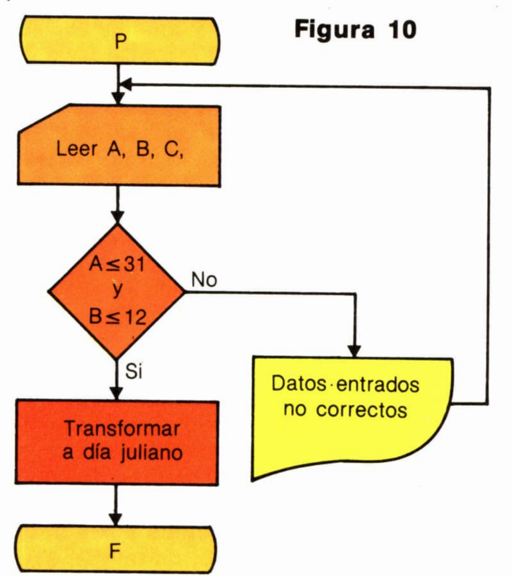


Figura 10

Para ilustrar estas explicaciones veamos unos cuantos ejemplos prácticos:

La condición $5 < 4$ es falsa y la condición $5 > 4$ es verdadera. Si creamos la condición compuesta $5 < 4$ o $5 > 4$, el resultado de ésta, según la tabla, será verdadero, ya que una de las dos es verdadera. Sin embargo, si creamos $5 < 4$ y $5 > 4$, el resultado será falso, ya que una de las dos es falsa. Aquí comprobamos con claridad que 5 no puede ser a la vez menor que 4 y mayor que 4; en la anterior también se ve claro que 5 o bien es menor que 4 o es mayor que 4, porque no puede ser igual. Por ejemplo «$4 > 0$ y $4 > 2$» es verdadera porque las dos condiciones se cumplen y $4 < 0$ o $4 < 2$ es falsa porque ninguna de ellas es verdadera.

Los operadores lógicos en las condiciones compuestas

Supongamos que vamos a construir un diagrama porque queremos que la computadora nos distinga si los datos del día y el mes de nacimiento de una serie de personas, que se están introduciendo en la computadora para procesarlos, se entran correctamente o no.

Si nos fijamos en el diagrama de la **figura 9**, veremos que se leen tres datos, el día (A), el mes (B) y el año (C), y que después se comprueba si el día es menor o igual a 31. Si no se cumple la condición, se imprime un mensaje indicando datos incorrectos y se vuelve a leer de nuevo. Si la condición se cumple, seguimos y se comprueba si el mes es menor o igual a 12. Si no se cumple esta condición, también se imprime el mismo mensaje anterior y se leen de nuevo los datos. Si se cumple, quiere decir que los datos son correctos y que están listos para procesar, acción que se realizará a continuación, ya que a partir de ellos se calcula el día del calendario juliano. Este cálculo sirve para saber una fecha determinada dentro de los 365 días del año. Por ejemplo, el 1 de enero de 1965 era el día 1 y el día 30 de abril de 1985 era el día 120 de los 365 de que consta el año.

El diagrama que hemos comentado tiene dos bloques de decisión contiguos con condiciones sencillas. Las dos condiciones simples son $A \leqslant 31$ y $B \leqslant 12$; como podemos comprobar, si no se cumple la primera condición, el diagrama se dirige al mismo lugar que si no se cumple

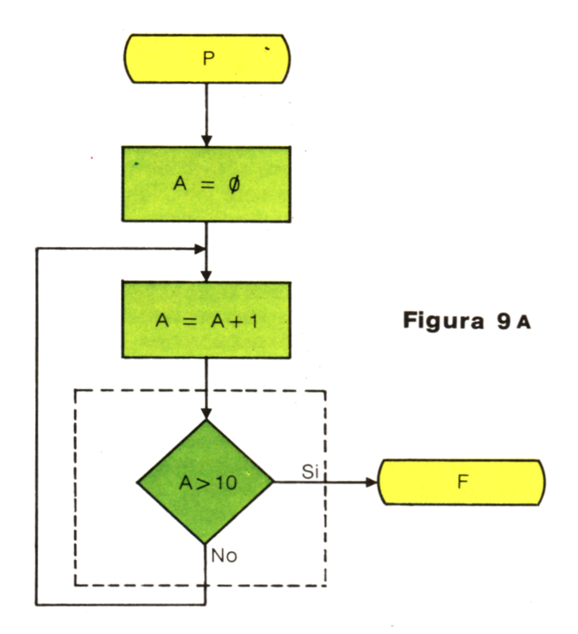


Figura 9 A

Figura 10 A

la segunda. Si se cumple la primera pero no se cumple la segunda, también va al mismo lugar en el que se imprime un mensaje de error. Solamente se sigue en secuencia y, por lo tanto, se dan como válidos los resultados entrados cuando las dos condiciones se cumplen. Observaremos que esta estructura es igual a la que tenemos en la **tabla de resultados de los operadores lógicos** *(pág. 186)*.

Para efectuar la comparación analicemos el diagrama. Cuando la primera condición es verdadera y la segunda falsa, tal como se muestra en la primera línea de la tabla, el resultado en el diagrama es falso o no correcto, porque se imprime un mensaje de error. Si en la tabla buscamos el resultado F de esta primera combinación, lo encontraremos bajo A y B y bajo NO A. Si seguimos analizando la segunda combinación y nos fijamos en el diagrama, si la primera condición es F, ya no preguntamos por la segunda, porque no importa que ésta sea verdadera o falsa; el resultado siempre es el mismo: falso. Si en dicha tabla buscamos el resultado de las dos combinaciones F V y F F, que es falso, lo encontramos bajo A y B; finalmente, al analizar la última combinación en que las dos son verdaderas, el resultado en el diagrama es verdadero y, en la tabla citada, este resultado también se encuentra bajo A y B.

De todo ello se deduce que podemos reducir estas dos condiciones simples a una sola condición compuesta, tal como se muestra en la **figura 10**. Con la condición compuesta A $\leqslant$ 31 y B $\leqslant$ 12, se consiguen los mismos resultados que con las dos condiciones simples. De esta manera se simplifica la construcción del diagrama; no obstante, hay que tener en cuenta siempre la **tabla de resultados de los operadores lógicos**, porque para interpretar el diagrama es preciso tener el conocimiento de que tal condición compuesta se cumplirá solamente cuando las dos condiciones simples se cumplan.

Condiciones múltiples

Las condiciones múltiples son elementos de un diagrama de flujo donde pueden tomarse tantos caminos o decisiones como sea necesario, dependiendo del valor de una determinada expresión.

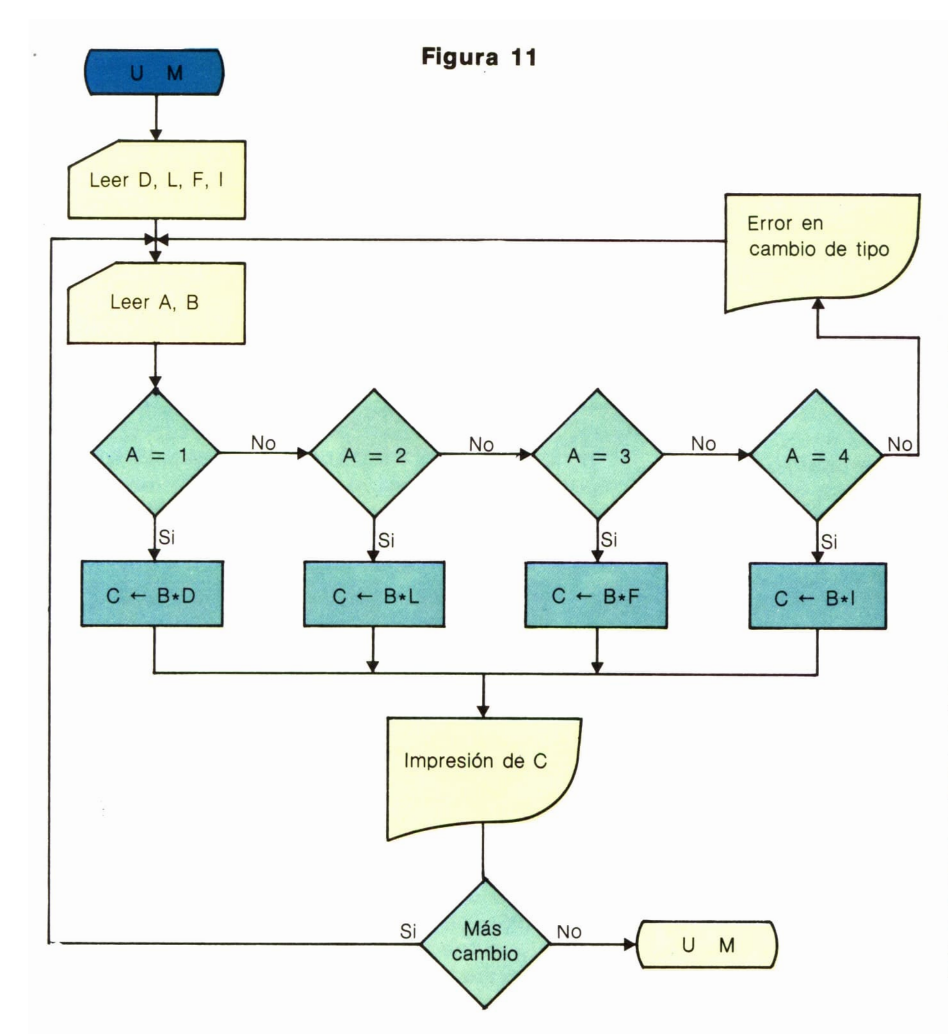
Se utilizan dentro de los diagramas y de los programas para sustituir a toda una serie de condiciones simples que no están unidas lógicamente.

Imaginemos que queremos cambiar moneda extranjera mediante la computadora. Solamente efectuaremos el cambio de dólares, liras, francos y libras a unidades monetarias (u. m.).

El diagrama de los pasos que hay que seguir con condiciones simples se muestra en la **figura 11** *(pág. 189)*.

Empezaremos leyendo el tipo de cambio del día, porque sufre cambios diarios. Entonces se leerá el número de u. m. que valdrá un dólar y se almacenará en D (de dólar); el número de u. m. que valdrá una lira y se almacenará en L (de lira); el número de u. m. que valdrá un franco y se almacenará en F (de franco); finalmente, el número de u. m. que valdrá una libra y se almacenará en I (de «inglés», porque la L ya la hemos utilizado en la lira).

A continuación leeremos el número de tipo de moneda que se quiere cambiar y cuánta moneda. El número significa que le asignaremos un número diferente a cada moneda, es decir, a los dólares les asignaremos moneda de tipo 1, a las liras el tipo 2, a los francos el tipo 3 y a las libras el tipo 4. El tipo lo asignaremos a A y la cantidad la almacenaremos en B, de tal manera que



Los diagramas de flujo constituyen el método más sencillo para la construcción de programas, y se realizan utilizando unos símbolos específicos. El diagrama de la izquierda corresponde a un caso con condiciones múltiples, es decir a un caso en el que pueden tomarse tantos caminos o decisiones como sea necesario en función del valor de una determinada expresión. El ejemplo elegido es el de decidir cambiar moneda extranjera. Se han considerado sólo cuatro monedas: el dólar, la lira, el franco y la libra. El primer paso es leer el valor de cambio en cada una de las divisas ese día, almacenándose el número de unidades monetarias que valdrá un dólar en D, el número de u.m. que valdrá una lira en L, el número de u.m. que valdrá un franco en F y el número de u.m. que valdrá una libra en I. Después de asociar un número a cada moneda (al dólar el 1, a la lira el 2, al franco el 3 y a la libra el 4) y consignarlo en A, almacenaremos en B la cantidad de moneda que se quiere cambiar. A continuación se construye una serie de condiciones simples, para averiguar el tipo de moneda que se quiere cambiar. La cantidad se multiplica entonces por el cambio, obteniéndose una salida impresa.

si leemos 1 y 100, significa que se quiere calcular cuántas unidades monetarias son 100 dólares, porque A = 1, que significa dólares, y B = 100, cantidad 100.

Una vez A y B han sido entradas, se construye una serie de condiciones simples, que más adelante veremos que se puede sustituir por una condición múltiple. Con estas condiciones se trata de averiguar qué tipo de moneda se quiere cambiar, ya que, como se ve en la primera condición, se pregunta por A = 1 (dólares), en la segunda por A = 2 (liras), en la tercera por A = 3 (francos), en la cuarta por A = 4 (libras) y, si A no vale ninguno de estos valores, se imprime un mensaje de error y se vuelve a leer el tipo de cambio y la cantidad.

Según el resultado de las condiciones, la cantidad de moneda se multiplica (recordemos que el signo * significa multiplicación) por el cambio de tal moneda, es decir por D si A = 1, por L si A = 2, por F si A = 3 o por I si A = 4. A continuación se pregunta si se quiere cambiar más moneda; si la respuesta es afirmativa, se vuelve a leer el tipo y la cantidad, si es que no finaliza el proceso.

Nótese que los valores D, L, F e I solamente se leen una vez, ya que su valor no cambia en todo el proceso.

Una vez explicado el funcionamiento del diagrama, veamos cómo podemos simplificarlo sustituyendo las condiciones simples por una múltiple. Esta simplificación se muestra en la **figura 12**, en la que se ha representado un bloque de decisión con cinco caminos posibles. El valor de la variable A (que es el tipo de cambio) dependerá de que se tome uno u otro camino.

LAS EXPRESIONES ARITMÉTICAS

En programación, una expresión aritmética es una combinación de variables y constantes unidas por operaciones aritméticas. Las operaciones aritméticas permitidas son las siguientes:

- La suma, representada por el signo +.
- La resta, representada por el signo −.
- La multiplicación, representada por el signo *.
- La división, representada por el signo /.
- La exponenciación, representada por el signo **.

En algunos lenguajes de programación, el signo ** es otro símbolo (^).

El diagrama de la izquierda corresponde a una simplificación del diagrama de flujo de la página anterior. En este caso se ha sustituido el conjunto de las cuatro condiciones simples (una para cada divisa) del diagrama construido, por una única condición múltiple, representada por un bloque de decisión con varios caminos posibles. La primera de las variables definidas, la A, que representa el tipo de cambio que se quiere realizar, tendrá un valor distinto según se tome uno u otro camino. En el programa escrito, la simplificación introducida equivale a sustituir cuatro instrucciones sucesivas por una única instrucción. El diafragma inferior muestra un ejemplo de inclusión de una expresión aritmética en un bloque de decisión y en una orden ejecutiva, simbolizados respectivamente por el rombo y el rectángulo.

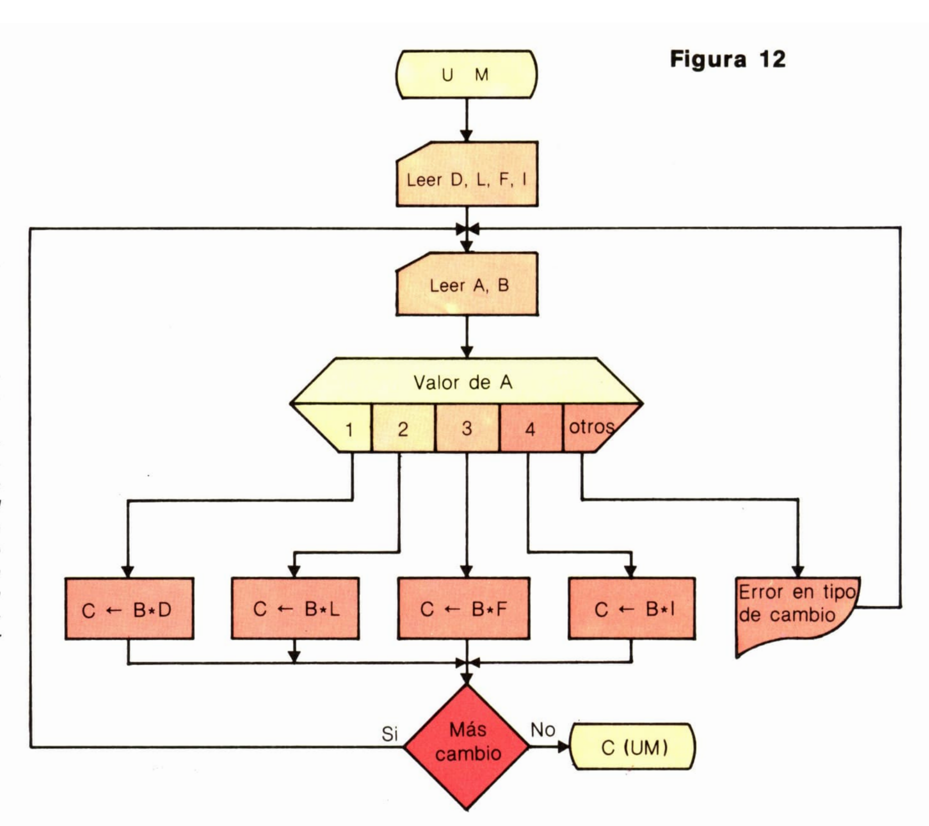


Figura 12

Normalmente, en los diagramas de flujo las expresiones aritméticas aparecen en los bloques de decisión o en las órdenes ejecutivas. Un ejemplo de cada una de ellas puede verse en la **figura 13**.

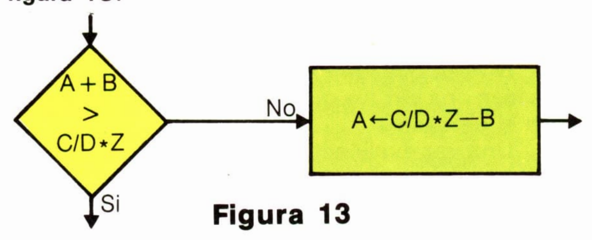


Figura 13

Fijémonos en la orden ejecutiva; en ella hay una asignación (←) sobre la variable A del resultado de una expresión aritmética. Esta expresión consta de 3 variables, 1 constante y 3 operaciones aritméticas; con ello tal y como se presenta, el problema consiste en saber qué operación se efectuará en primer lugar. Para resolver esta incógnita, se necesita saber qué prioridades existen por convenio en la ejecución de estas operaciones. Este orden de prioridades en programación es el mismo que el utilizado en álgebra (disciplina matemática uno de cuyos temas objeto de estudio es éste) y es el siguiente:

Prioridad	Operación aritmética
1	Exponenciación (**)
2	Multiplicación y división (*, /)
3	Suma y resta (+, −)

Es decir, primero se efectuarán todas las operaciones de exponenciación, después todas las multiplicaciones y divisiones y por último todas las sumas y las restas.

Para ilustrar este tema pongamos unos cuantos ejemplos. La expresión aritmética A + C * D, si no hubiera ninguna regla, podría interpretarse de dos maneras diferentes:

- sumar A con C y multiplicar el resultado de la suma por D;
- multiplicar C por D y sumar el resultado con A.

El resultado de estas dos maneras de resolver la expresión aritmética es diferente. Veámoslo. Supongamos que A = 5, C = 3 y D = 2, entonces la expresión será 5 + 3 * 2.

El resultado de la primera interpretación es 16, porque sumaríamos 5 y 3, que daría 8 y este resultado lo multiplicaríamos por 2, con lo que nos quedaría 16.

Sin embargo, aplicando la segunda interpretación, multiplicaríamos primero 3 por 2, que daría 6, y este resultado lo sumaríamos a 5, con lo que obtendríamos 11. Por lo tanto, hemos comprobado que un caso nos da 16 y el otro 11.

El modo correcto de resolverlo es siguiendo el orden de prioridades descrito antes, que coincide con la segunda manera de hacerlo.

Uno de los problemas que puede presentarse al aplicar esta norma es el de dos operaciones que tengan la misma prioridad. Por ejemplo, en

la expresión A * B / C. En este tipo de expresiones el álgebra no es muy explícita; sin embargo, en programación la regla es muy clara: «La ejecución entre operaciones de la misma prioridad siempre se efectuará de izquierda a derecha.»

Si no aplicáramos esta regla, la expresión anterior podría interpretarse como $\frac{A * B}{C}$ o como $A * \frac{B}{C}$; sin embargo, aplicando la regla se interpreta como $\frac{A * B}{C}$, es decir, primero la multiplicación, porque está la primera de izquierda a derecha, y después la división.

No obstante, para no tener que aplicar esta regla en cada caso, podemos optar por el empleo del paréntesis, de modo que lo que está entre paréntesis siempre es lo que tiene más prioridad. Si los dos ejemplos anteriores A + C * D y A * B / C los expresábamos de la forma (A + C) * D y A * (B / C), en estos casos se ejecutaría antes la operación que se encuentra dentro del paréntesis y después la de fuera. También podrían expresarse de la forma A + (C * D) y (A * B) / C, con lo cual estaríamos indicando lo mismo que A + C * D y A * B / C; la única diferencia es que en los dos casos con paréntesis aplicaríamos la regla del paréntesis y en los dos sin paréntesis aplicaríamos la regla de las prioridades, pero los resultados serían los mismos.

Veamos un caso en que aparezcan todas las operaciones posibles:

A*B−C+D**A / E

- primero efectuaríamos la exponenciación, que podemos expresarla así A*B−C+ +(D**A) / E;
- en segundo lugar efectuaríamos la multiplicación: (A*B)−C+(D**A) / E;
- en tercer lugar realizaríamos la división, que podríamos expresarla así: (A*B)− −C+((D**A) / E);
- en cuarto lugar haríamos la resta y lo podríamos expresar así: ((A*B)−C)+ ((D**A) / E);
- por lo tanto, en último lugar haríamos la suma.

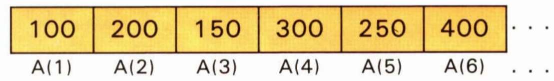
La expresión A*B−C+D**A / E también se puede indicar de la forma ((A*B)−C)+((D**A)/ /E); en este caso se ejecuta sin necesidad de aplicar la regla de las prioridades, ya que se utilizan paréntesis.

VARIABLES CON ÍNDICE

Tal como hemos explicado en un apartado anterior, las variables sirven para almacenar valores que pueden ir cambiando, pero que siempre, en un momento determinado, almacenan un solo valor, es decir, una misma variable, por ejemplo A: si su valor es 5, no puede valer al mismo tiempo 4. Sin embargo, a veces conviene expresar mediante el mismo nombre de variable valores numéricos diferentes.

Para una mejor comprensión, veamos un ejemplo. Supongamos que tenemos toda una serie de valores y los queremos almacenar uno junto a otro, porque todos guardan relación. La mejor manera de tenerlos almacenados es en un vector. Este vector puede ser tan largo como se quiera, es decir, puede tener tantos elementos como se precisen.

Figura 14



Si nos fijamos en la **figura 14**, podemos ver un ejemplo de un vector con toda una serie de elementos que son los valores que hemos almacenado en él. A dicho vector le hemos llamado A; es decir, cuando nos referimos a A, nos estamos refiriendo con esta variable a todos los elementos del vector o a todos los valores almacenados en él. Ahora bien, si queremos referirnos a un elemento determinado del vector, entonces deberemos utilizar el índice. Por ejemplo, para hacer referencia al valor 250 del vector de la **figura 14**, lo haremos mediante la variable A, que identifica a todo el vector, y el índice (5), que identifica al quinto elemento del vector. Por lo tanto, A(5) = 250. De esta misma manera identificaremos a cada elemento del vector:

A(1) = 100
A(2) = 200
A(3) = 150
A(4) = 300
A(5) = 250
A(6) = 400

Para expresar un elemento del vector, podemos tener dos variables: una que identifique a todo el vector y otra que identifique al índice. Así por ejemplo, al especificar A(I), estamos identificando un elemento del vector que estará en la posición indicada por I. Por ejemplo, si I = 2, entonces nos estaremos refiriendo al segundo elemento del vector y así sucesivamente, en el caso de que el vector tuviera más de seis elementos.

Cuando necesitamos trabajar con un solo vector en un diagrama, utilizamos siempre bucles sencillos. Para entender este tratamiento, ver el apartado «Bucles sencillos» (pág. siguiente).

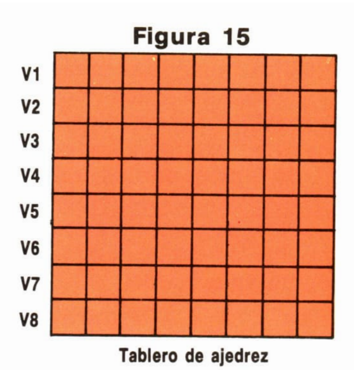
Hasta ahora hemos estado hablando de cómo es un vector, pero no hemos mencionado para nada cómo es o cómo se llama un grupo o conjunto de vectores, tema que describiremos a continuación.

A un conjunto de vectores se le denomina matriz; ésta puede ser de dos dimensiones, de tres, de cuatro, etcétera.

En principio, hablemos de la de dos dimen-

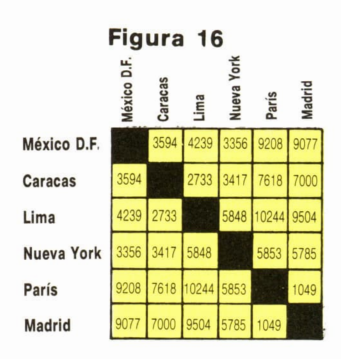
Las matrices son conjuntos de vectores, y pueden ser de dos, tres, o más dimensiones. En el dibujo se ilustra una matriz bidimensional formada por ocho vectores de ocho elementos. La matriz tendrá por lo tanto 64 elementos, igual número de cuadrados que tiene un tablero de ajedrez.

Figura 15



En la matriz del dibujo en cada casilla figuraría la distancia aérea en kilómetros entre dos ciudades: las indicadas por las líneas horizontal y vertical. En las casillas negras de la diagonal figurarían ceros y las cifras que aparecerían en casillas simétricamente situadas respecto a esa diagonal serían iguales.

Figura 16

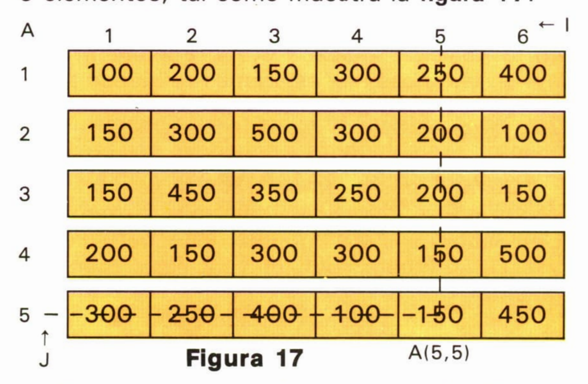


	México D.F.	Caracas	Lima	Nueva York	París	Madrid
México D.F.		3594	4239	3356	9208	9077
Caracas	3594		2733	3417	7618	7000
Lima	4239	2733		5848	10244	9504
Nueva York	3356	3417	5848		5853	5785
París	9208	7618	10244	5853		1049
Madrid	9077	7000	9504	5785	1049	

siones mediante un ejemplo: un tablero de ajedrez es una matriz de dos dimensiones, formada por ocho vectores de ocho elementos cada uno.

Si nos fijamos en la **figura 15**, podemos ver esta circunstancia. V1 es el primer vector y tiene 8 elementos, V2 es el segundo y así hasta V8, que es el último; el total de elementos de la matriz es de 64, resultado de multiplicar 8 elementos de cada vector por los 8 vectores existentes. Lógicamente, en estas matrices cada elemento tendrá su valor. Otro ejemplo práctico es la matriz dibujada en la **figura 16**; se trata del cálculo de la distancia aérea en kilómetros entre varias ciudades importantes. Para poder buscar la distancia kilométrica entre México D. F. y Nueva York, buscaremos primero horizontalmente el nombre de la primera ciudad y verticalmente el nombre de la segunda. Una vez encontradas, trazaremos una línea vertical hacia abajo desde la posición horizontal encontrada y una línea horizontal hacia la derecha desde la posición vertical encontrada; allí donde se encuentren las dos líneas estará el dato que estamos buscando, es decir, la distancia kilométrica entre las dos ciudades.

Podríamos también volver sobre el ejemplo de la **figura 14** y suponer que, en lugar de un vector de 6 elementos, tenemos 5 vectores de 6 elementos, tal como muestra la **figura 17**.



A	1	2	3	4	5	6 ← I
1	100	200	150	300	250	400
2	150	300	500	300	200	100
3	150	450	350	250	200	150
4	200	150	300	300	150	500
5	300	250	400	100	150	450

Figura 17

En este caso también vamos a identificar a todo el conjunto de vectores con la variable A; sin embargo, para identificar cada elemento ya no podemos hacerlo con un solo índice, sino que necesitamos dos. Por lo tanto, definiremos dos variables más para los dos índices. Supongamos que sean I para indicar el índice de las columnas y J para el índice de las filas. El índice I podrá tener valores del 1 al 6 y el índice J podrá tener valores del 1 al 5. Adoptando estas variables, para indicar cualquier elemento de la matriz se hará de la forma A(I,J), es decir, la variable A para indicar toda la matriz, el índice I para indicar en qué columna se encuentra el elemento y el índice J para indicar en qué fila se encuentra.

Por ejemplo, el elemento A(5,5) será el correspondiente a la columna 5 y a la fila 5. Se traza desde esta columna una línea vertical hacia abajo y, al mismo tiempo, una línea horizontal a la derecha desde la fila indicada; la posición donde se encuentren esas dos líneas será el elemento al que nos estamos refiriendo.

Todos estos casos aluden a las matrices de dos dimensiones; para describir las matrices de más de dos dimensiones, sería mucho más difícil y complicado; pero lo que se pretende es que se comprenda el concepto de matriz. Sin embargo, digamos solamente que una matriz de tres dimensiones es la que tiene tres índices, la de cuatro dimensiones la que tiene cuatro índices y así sucesivamente.

Cuando necesitamos trabajar con matrices normalmente en diagrama, utilizamos siempre bucles anidados. Este tratamiento se explica en el apartado «Bucles anidados» *(pág. 196)*.

LOS BUCLES

El bucle es una estructura que se usa frecuentemente en programación y que es muy útil para evitar repeticiones innecesarias.

Consta siempre de una serie de pasos, cuya ejecución se repite sucesivamente hasta que se cumple una determinada condición; ésta hace que la secuencia de ejecución continúe con los pasos siguientes a los del bucle.

Existen dos categorías de bucles: los bucles sencillos y los bucles anidados.

Bucles sencillos

De la descripción de lo que es un bucle se deduce que su estructura más sencilla siempre constará, como mínimo, de una sentencia ejecutiva y de un bloque de decisión, en el cual se

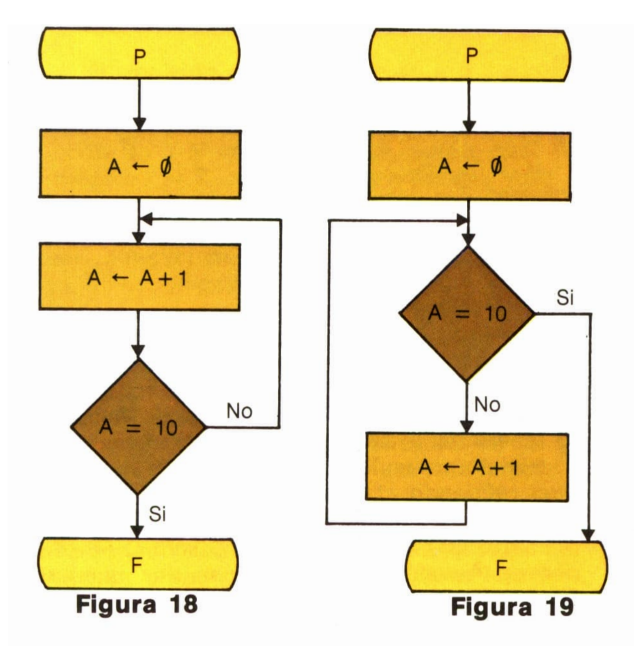
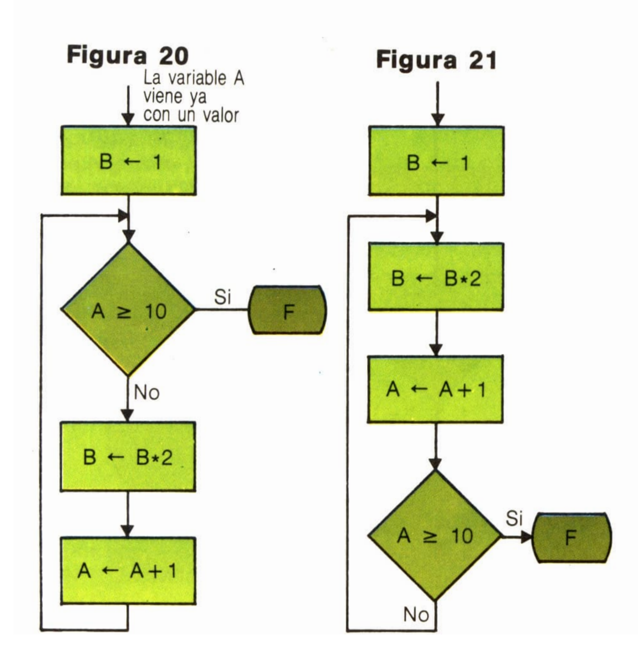


Figura 18 **Figura 19**

comprobará si la condición mediante la cual se decide terminar con la ejecución del bucle se cumple o no. En la **figura 18** se muestra el bucle más sencillo que puede construirse, formado por la sentencia ejecutiva $A \leftarrow A + 1$, y el bloque de decisión cuya condición es $A = 10$.

Este bucle es simplemente un contador de interacciones que va aumentando el valor de una variable (A) hasta que éste sea 10, momento en el que se termina la ejecución del bucle y finaliza el diagrama. El mismo bucle de la **figura 18** lo podemos ver construido de otra manera en la **figura 19** y, sin embargo, realizan la misma función; sólo que uno incrementa el valor de la variable antes de preguntar por la condición y el otro pregunta y después incrementa.

El hecho de que estos dos bucles hagan lo mismo, a pesar de estar construidos de diferente manera, se debe a que solamente se efectúan en ellos el incremento de la variable y el bloque de decisión.


Figura 20 **Figura 21**

Veamos un ejemplo en el que la variable A no tenga que empezar obligatoriamente de cero y veremos que, en este caso, los dos bucles construidos de diferente manera no realizan lo mismo.

Si nos fijamos en las **figuras 20** y **21**, veremos la parte final de dos diagramas de flujo que está compuesta en cada uno de ellos por un mismo bucle, pero construido de manera diferente. Al ser la parte final, el valor de la variable A, cuando la ejecución llega al bucle, puede que sea 10 o mayor, porque esta variable ha tomado ya algún valor al ejecutarse la parte anterior del diagrama. Suponiendo que A venga con el valor 12, ¿qué sucede en cada uno de estos diagramas? En el de la **figura 20** la variable B tomará valor 1 e inmediatamente se comprobará si A es $\geqslant$ a 10. Al ser A igual a 12, entonces la condición se cumplirá y habremos finalizado la ejecución con las variables $A = 12$ y $B = 1$. Sin embargo, en el diagrama de la **figura 21** la variable B tomará valor 1 e inmediatamente después B tomará el valor 2, porque se le asignará $B * 2$; como $B = 1$, $1 * 2$ será igual a 2. Seguidamente, A se incrementará en 1 y pasará a valer 13, pasándose después a comprobar si la condición $A \geqslant 10$ se cumple; como efectivamente se cumple, se habrá finalizado la ejecución con las variables $A = 13$ y $B = 2$.

Por lo tanto, el diagrama de la **figura 20** no produce los mismos resultados que el de la **figura 21**, ya que las variables A y B de cada uno de ellos no tienen el mismo valor; esto es debido a la construcción diferente de los dos bucles.

De ahí que hay que prestar mucha atención a la construcción de un bucle y que la manera de construirlo puede alterar los resultados esperados producidos por la ejecución del programa que resultará de la codificación del diagrama.

En todos los bucles que hemos visto y en los que interviene una variable que actúa como contador, esta variable sufre incrementos positivos, lo cual no quiere decir que no pueda sufrir incrementos negativos, ya que en este caso se comportará como contador igualmente.

Por ejemplo, en el bucle de la **figura 18** se utiliza la variable A como contador y los incrementos que sufre son positivos; sin embargo, podría sufrir incrementos negativos y realizar la misma función. Esto puede verse en la **figura 22**. El incremento que sufre la variable A es de -1 en -1, con lo que irá siendo cada vez menor o más negativa. La condición de salida del bucle, lógicamente, no podrá ser $A = 10$, porque, si A va sufriendo incrementos negativos, nunca llegará a valer 10 y por lo tanto resultará un bucle cuya ejecución no terminará nunca. La condición, en este caso, será $A = -10$, con lo cual el bucle se ejecutará también 10 veces.

IN

Figura 23

17	17	16	18	17	18	16	17	17	17
1	2	3	4	5	6	7	8	9	10

Alumno 1	17 años	Alumno 6	18 años
Alumno 2	17 años	Alumno 7	16 años
Alumno 3	16 años	Alumno 8	17 años
Alumno 4	18 años	Alumno 9	17 años
Alumno 5	17 años	Alumno 10	17 años

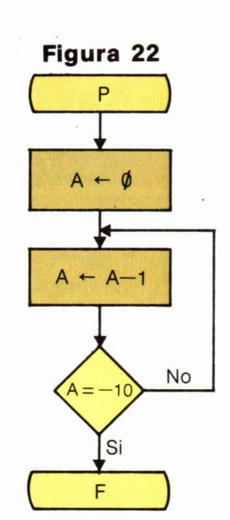
Uso de bucles sencillos para el tratamiento de vectores

Como ya hemos explicado en el apartado de variables con índice, para trabajar con vectores siempre utilizamos los bucles sencillos. Esto se hace porque siempre, dentro de un vector, hay que ir accediendo a cada uno de sus elementos al realizar algún trabajo con él. Este acceso normalmente es secuencial, es decir, se accede al primer elemento, después al segundo, al tercero y así sucesivamente hasta el último. Por ejemplo, consideremos el vector de la **figura 23**, el cual contiene las edades de 10 alumnos de un curso de inglés.

Al vector le llamamos IN, por lo tanto, cuando nos referimos a IN, estamos hablando de los 10 alumnos de inglés. Ahora bien, interesaría saber exactamente la edad media de todos estos alumnos. Sumaremos todos los elementos del vector y dividiremos el resultado por el número de alumnos, en este caso, 10.

Es imprescindible, por lo tanto, el acceso secuencial a cada uno de los elementos del vector para irlos sumando y, como ya hemos dicho, la mejor manera de efectuar este acceso es un bucle sencillo. Para ver cómo se representa este proceso, se ha construido el diagrama de la **figura 24**, en el cual se ve que en principio se leen las edades de los alumnos y se almacenan en el vector IN, tal como se ve en la **figura 23**. Una vez almacenadas, ya tenemos el vector listo para su tratamiento.

El primer paso consiste en acceder al primer elemento del vector. Esto se consigue asignando un 1 a la variable I y especificando IN(I). Esto quiere decir que, como I = 1, entonces IN(I) se transforma en IN(1) y, por lo tanto, nos estamos refiriendo al primer elemento del vector. Con este elemento se efectúa la expresión A + IN(I), que en el primer caso será 0 + IN(1), porque A valdrá 0; con lo que nos quedará 0 + 17 y este resultado se le asignará a A, por lo tanto A = 17. Como vemos, A actuará de acumulador y es donde quedará el resultado de la suma de todas las edades.



Los bucles se utilizan en programación para evitar repeticiones innecesarias. Implican la repetición de una serie de instrucciones hasta que se cumpla determinada condición. El que se ilustra en el diagrama adjunto es similar al de la figura 18, sólo que en aquel se consideraba que la variable A sufría incrementos positivos de unidad en unidad, mientras que en éste se supone que se incrementa de —1 en —1. El bloque de decisión ordenará la repetición de la operación hasta que A llegue a valer —10, momento en que cesará la iteración.

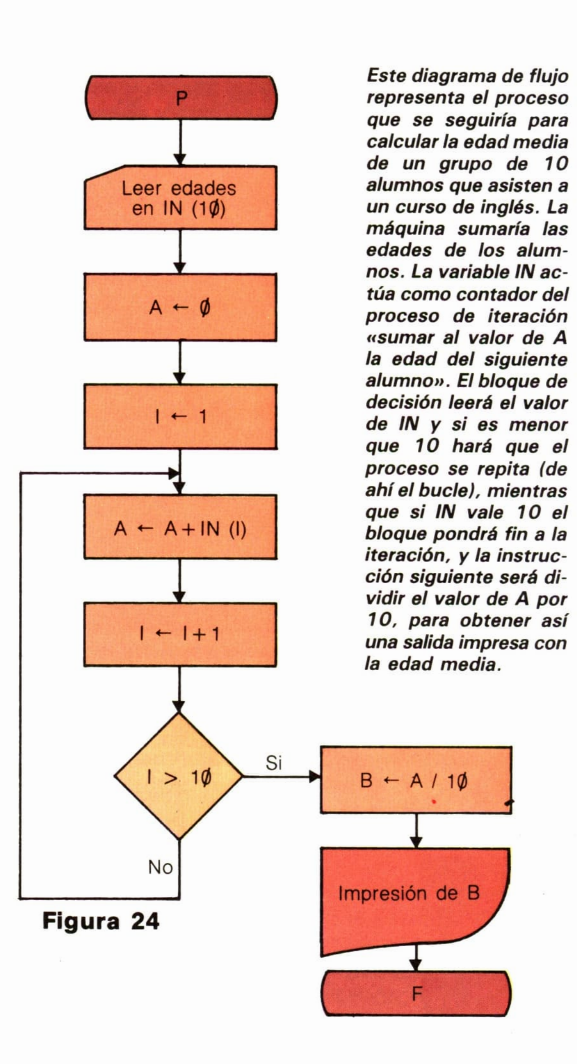


Figura 24

Este diagrama de flujo representa el proceso que se seguiría para calcular la edad media de un grupo de 10 alumnos que asisten a un curso de inglés. La máquina sumaría las edades de los alumnos. La variable IN actúa como contador del proceso de iteración «sumar al valor de A la edad del siguiente alumno». El bloque de decisión leerá el valor de IN y si es menor que 10 hará que el proceso se repita (de ahí el bucle), mientras que si IN vale 10 el bloque pondrá fin a la iteración, y la instrucción siguiente será dividir el valor de A por 10, para obtener así una salida impresa con la edad media.

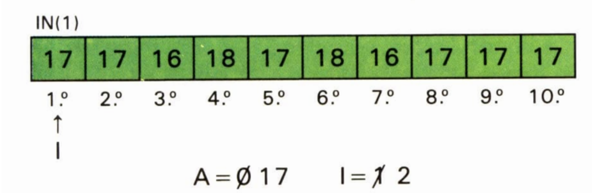
El siguiente paso consiste en ir a buscar el segundo elemento del vector. Esto se consigue con la sentencia I ← I+1; es decir, se incrementa en 1 el valor del índice, con lo cual nos quedará I = 2, porque hasta entonces valía 1. La siguiente sentencia es el bloque de decisión típico de todo bucle. Se comprueba si I (el índice) es mayor de 10. Si lo es, quiere decir que ya hemos llegado al final del vector y, por lo tanto, ya tenemos calculada la suma; luego se puede calcular la media, cosa que se hace en la sentencia B ← A / 10. Como I = 2 no se cumplirá la condición; el bucle nos conducirá a la repetición de la suma de A y el segundo elemento IN(2); así se irá repitiendo la ejecución de las sentencias del bucle hasta que se cumpla la condición.

Como ya hemos dicho, el resultado de la suma de todos los elementos del vector lo tendremos en la variable A y, para calcular la media, lo dividiremos por el número de elementos o alumnos, que será 10. El resultado almacenado en A será 17+17+16+18+17+18+16+ +17+17+17 = 170 y, por lo tanto, la media de edades será 17, ya que 170 / 10 = 17. Para verlo más claro, vamos a hacer una descripción del seguimiento paso a paso del diagrama, mostrando incluso el elemento del vector al que se va accediendo en cada paso.

• Antes de empezar el bucle,

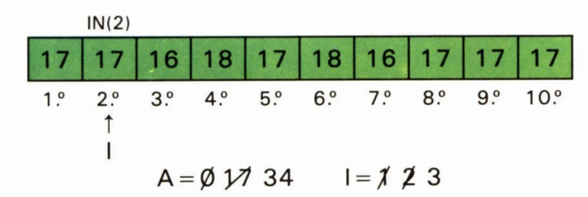
A = 0   I = 1

• *Primera pasada por el bucle*. Nos posicionamos en IN(I) y, como I = 1, en el elemento IN(1),



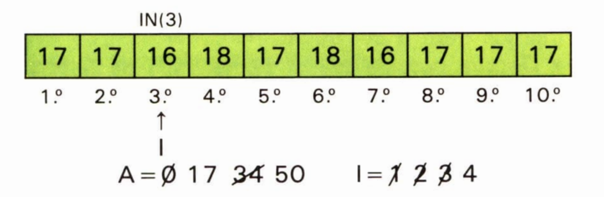
Como el valor de I no es mayor de 10, seguimos dentro del bucle.

• *Segunda pasada por el bucle*. Nos posicionamos en IN(I) y, como I = 2, en el elemento IN(2),



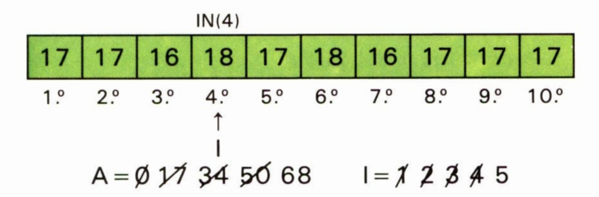
Como el valor de I no es mayor de 10, seguimos dentro del bucle.

• *Tercera pasada por el bucle*. Nos posicionamos en IN(I) y, como I = 3, en el elemento IN(3),



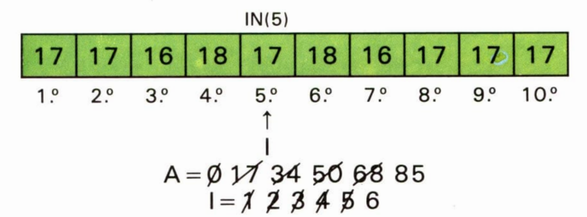
Como el valor de I no es mayor de 10, seguimos dentro del bucle.

• *Cuarta pasada por el bucle*. Nos posicionamos en IN(I) y, como I = 4, en el elemento IN(4),



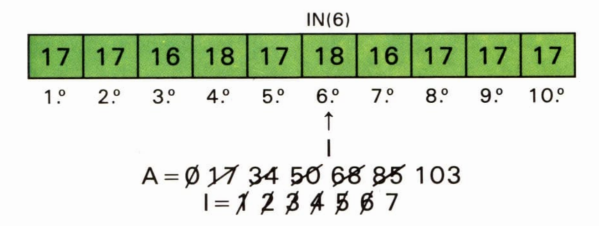
Como el valor de I no es mayor de 10, seguimos dentro del bucle.

• *Quinta pasada por el bucle*. Nos posicionamos en IN(I) y, como I = 5, en el elemento IN(5),



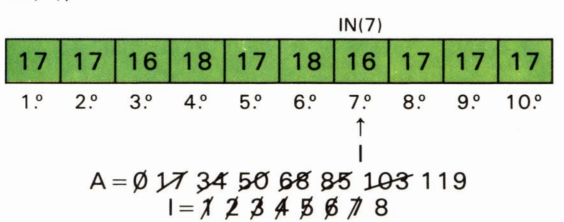
Como el valor de I no es mayor de 10, seguimos dentro del bucle.

• *Sexta pasada por el bucle*. Nos posicionamos en IN(I) y, como I = 6, en el elemento IN(6),



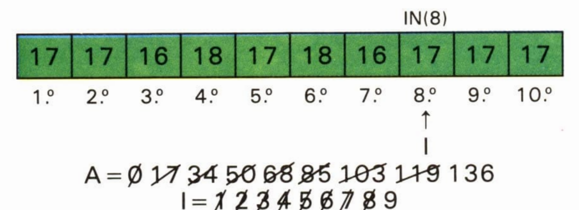
Como el valor de I no es mayor de 10, seguimos dentro del bucle.

• *Séptima pasada por el bucle*. Nos posicionamos en IN(I) y, como I = 7, en el elemento IN(7),



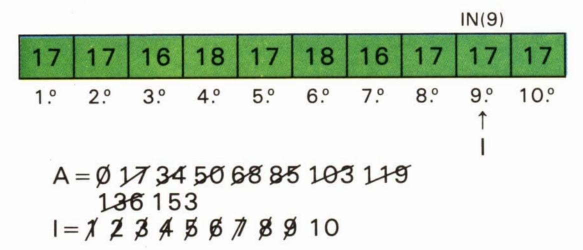
Como el valor de I no es mayor de 10, seguimos dentro del bucle.

• *Octava pasada por el bucle*. Nos posicionamos en IN(I) y, como I = 8, en el elemento IN(8),



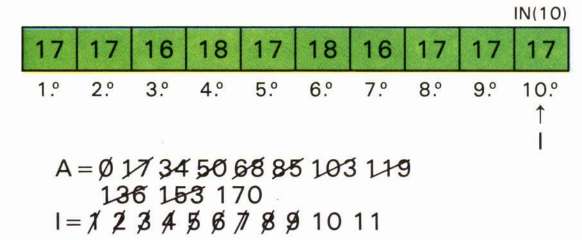
Como el valor de I no es mayor de 10, seguimos dentro del bucle.

• *Novena pasada por el bucle*. Nos posicionamos en IN(I) y, como I = 9, en el elemento IN(9),



Como el valor de I no es mayor de 10, seguimos dentro del bucle.

• *Décima pasada por el bucle*. Nos posicionamos en IN(I) y, como I = 10, en el elemento IN(10),



El valor de I es mayor de 10, con lo que se cumple la condición para salir del bucle. Salimos y calculamos la media.

B = A / 10; como A = 170,

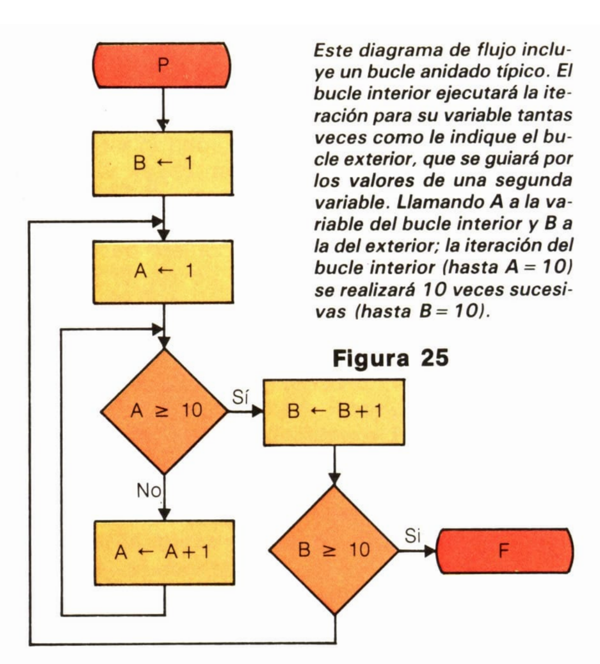
B = 170 / 10 y B = 17

El diagrama está bien construido.

Como vemos, el bucle sencillo que hay en el diagrama de la **figura 26** nos ha servido perfectamente para trabajar y movernos por los elementos del vector que habíamos construido.

Bucles anidados

Nos encontraremos con estructuras de bucles anidados cuando tengamos que construir un bucle dentro de otro o dentro de varios. La necesidad de la construcción de estos bucles es difícil de comprender, ya que normalmente responden a procedimientos de cálculo un poco complejos.



Este diagrama de flujo incluye un bucle anidado típico. El bucle interior ejecutará la iteración para su variable tantas veces como le indique el bucle exterior, que se guiará por los valores de una segunda variable. Llamando A a la variable del bucle interior y B a la del exterior; la iteración del bucle interior (hasta A = 10) se realizará 10 veces sucesivas (hasta B = 10).

Figura 25

Para entender mejor su funcionamiento, explicaremos primero mediante un diagrama cómo se construyen los bucles anidados y después describiremos un caso práctico.

Para ver un diagrama típico que contiene un bucle anidado, fijémonos en la **figura 25**.

El diagrama de flujo de esta figura es todo él un bucle anidado, donde el bucle interior se ejecuta tantas veces como indique el bucle exterior. Es decir, al ejecutar tal diagrama, primero se entrará en el bucle interior y se ejecutará, lo cual quiere decir que se ejecutará 10 veces, porque empezará con A ← 1 y no terminará hasta que A sea ⩽ 10. Una vez se haya cumplido la condición, se pasará al bucle exterior, se incrementará el valor de B, se examinará si se cumple la condición de este bucle exterior y, si no se cumple, se volverá a ejecutar el bucle interior 10 veces de nuevo, porque el valor de A empezará otra vez por 1.

Si siguiéramos paso a paso este diagrama (cosa que no haremos aquí, sino más adelante con un caso real), veríamos que, por cada vez que ejecutamos el bucle exterior, se ejecuta 10 veces el bucle interior.

Una representación de lo que sucedería siguiendo paso a paso con el valor de las variables sería:

A = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ B = ~~1~~
A = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ B = ~~2~~
A = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ B = ~~3~~
A = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ B = ~~4~~
A = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ B = ~~5~~
A = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ B = ~~6~~
A = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ B = ~~7~~
A = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ B = ~~8~~
A = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ B = ~~9~~
A = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ 10 B = 10

Si intentásemos realizar el seguimiento paso a paso, éstos serían los valores de las variables correspondientes a cada bucle.

Ahondando en lo que decíamos antes, vemos que, mientras la variable A (que pertenece al bucle exterior) aumenta su valor en 1 unidad, la variable A (que pertenece al bucle interior) ha aumentado de 1 en 1 hasta 10 y vuelve a empezar por 1.

Los bucles anidados tienen que adoptar esta representación para estar bien construidos y ser considerados correctos; en cambio, esta construcción no es correcta. O sea, siempre habrá un bucle sencillo metido completamente en el otro y no solamente una parte.

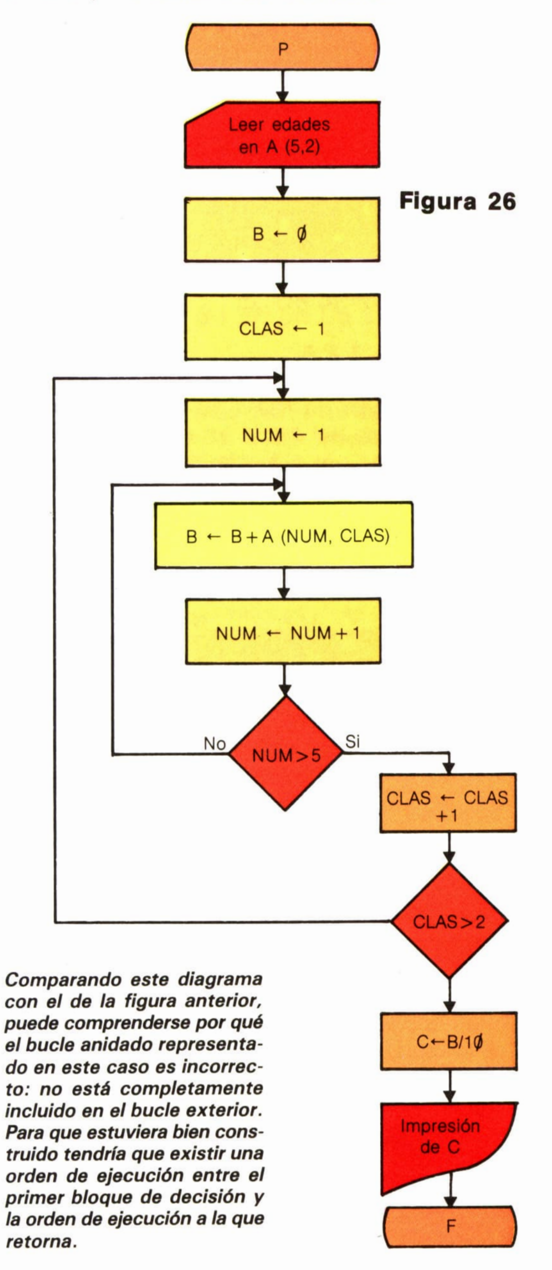


Figura 26

Comparando este diagrama con el de la figura anterior, puede comprenderse por qué el bucle anidado representado en este caso es incorrecto: no está completamente incluido en el bucle exterior. Para que estuviera bien construido tendría que existir una orden de ejecución entre el primer bloque de decisión y la orden de ejecución a la que retorna.

Uso de bucles anidados para el tratamiento de matrices

Para trabajar con matrices, siempre se utilizan los bucles anidados. Para ilustrar esta explicación hemos elegido el mismo caso práctico citado en el apartado «Uso de bucles sencillos para el tratamiento de vectores» *(pág. **194**)*, es decir, una clase de inglés formada por 10 alumnos; pero, en este caso, en lugar de una clase serán dos clases y en lugar de 10 alumnos serán 5. Por lo tanto, tendremos dos vectores con las edades, en cada uno de ellos, de los alumnos de cada clase de inglés, tal como se muestra a continuación.

CLASE 1		CLASE 2	
alumno 1	17 años	alumno 1	18 años
alumno 2	17 años	alumno 2	18 años
alumno 3	16 años	alumno 3	16 años
alumno 4	18 años	alumno 4	17 años
alumno 5	17 años	alumno 5	16 años

CLASE 1	17	17	16	18	17
CLASE 2	18	18	16	17	16

En este tipo de representación tenemos dos vectores que formarán una matriz de dos dimensiones. A la matriz la llamaremos A y a los índices les llamaremos NUM para las columnas (de n.º de alumno) y CLAS para las filas (de clase). Entonces, para referirnos a un elemento utilizaremos la estructura A(NUM, CLAS). Por ejemplo, cuando NUM = 5 y CLAS = 2, nos referiremos al alumno número 5 de la clase 2 o, lo que es lo mismo, A(5, 2).

Supongamos ahora que queremos también calcular la media de edad de todos los alumnos de las dos clases. Entonces, para realizar este cálculo, necesitaremos tener acceso a todas las edades de los alumnos y, como las vamos a tener en una matriz, para realizar el cálculo tendremos que construir el diagrama con un bucle anidado. Este diagrama se muestra en la **figura 26**.

Fijándonos un poco en lo que hace el diagrama, se ve cómo, al entrar en el primer bucle, el interior, vamos accediendo a los elementos del primer vector, o sea, a las edades de los alumnos de la primera clase, ya que, mientras CLAS = 1, NUM va variando de 1 a 5; por lo tanto, se acumulan en B las edades de los alumnos de la primera clase, se termina el bucle interior, se accede al exterior y, como todavía no termina el exterior porque CLAS no es > 2, se vuelve a ejecutar el interior. Así que CLAS = 2 y NUM variará de 1 a 5, lo cual quiere decir que se irán acumulando en B las edades correspondientes a todos los alumnos de la clase 2; hasta que, al terminar el bucle interior, también se

termina el exterior. Con ello se tiene en la variable B la suma de todas las edades de los alumnos de las dos clases; se calculará la media de todos los alumnos, dividiendo por 10, ya que éste es el número total de alumnos.

Calculándolo por simple adición sería:

$$B = 17 + 17 + 16 + 18 + 17 + 18 + 18 + 16 + 17 + 16 = 170$$

$B = 170$ $\quad C = 170/10 \quad$ $C = 17$

Llegamos a que C = 17, es decir, la media será 17.

Para ver si hemos construido bien el diagrama y también para ver lo más detalladamente posible el funcionamiento del mismo, vamos a hacer una descripción del seguimiento paso a paso del diagrama, mostrando incluso el elemento de la matriz al que se va accediendo en cada paso.

Antes de empezar los bucles:

— B = 0;
— CLAS = 1;
— NUM = 1.

• *Primera pasada por el bucle exterior.*

• *Primera pasada por el bucle interior.* Nos posicionamos en A(NUM, CLAS) y, como NUM = 1 y CLAS = 1, en el elemento A(1, 1),

NUM ↓ (1.º)

	1.º	2.º	3.º	4.º	5.º
CLAS → 1.º	17	17	16	18	17
2.º	18	18	16	17	16

B = ~~0~~ 17
CLAS = 1
NUM = ~~1~~ 2

Como el valor de NUM no es mayor de 5, seguimos dentro del bucle interior.

• *Segunda pasada por el bucle interior.* Nos posicionamos en A(NUM, CLAS) y, como NUM = = 2 y CLAS = 1, en el elemento A(2, 1),

NUM ↓ (2.º)

	1.º	2.º	3.º	4.º	5.º
CLAS → 1.º	17	17	16	18	17
2.º	18	18	16	17	16

B = ~~0~~ ~~17~~ 34
CLAS = 1
NUM = ~~1~~ ~~2~~ 3

Como el valor de NUM no es mayor de 5, seguimos dentro del bucle interior.

• *Tercera pasada por el bucle interior.* Nos posicionamos en A(NUM, CLAS) y, como NUM = = 3 y CLAS = 1, en el elemento A(3, 1),

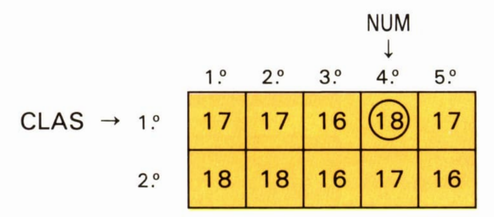
NUM ↓ (3.º)

	1.º	2.º	3.º	4.º	5.º
CLAS → 1.º	17	17	16	18	17
2.º	18	18	16	17	16

B = ~~0~~ ~~17~~ ~~34~~ 50
CLAS = 1
NUM = ~~1~~ ~~2~~ ~~3~~ 4

Como el valor de NUM no es mayor de 5, seguimos dentro del bucle interior.

• *Cuarta pasada por el bucle interior.* Nos posicionamos en A(NUM, CLAS) y, como NUM = = 4 y CLAS = 1, en el elemento A(4, 1),



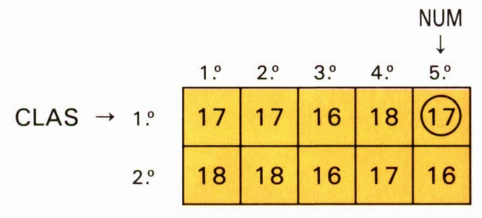
NUM ↓ (4.º)

	1.º	2.º	3.º	4.º	5.º
CLAS → 1.º	17	17	16	18	17
2.º	18	18	16	17	16

B = ~~0~~ ~~17~~ ~~34~~ ~~50~~ 68
CLAS = 1
NUM = ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

Como el valor de NUM no es mayor de 5, seguimos dentro del bucle interior.

• *Quinta pasada por el bucle interior.* Nos posicionamos en A(NUM, CLAS) y, como NUM = = 5 y CLAS = 1, en el elemento A(5, 1),



NUM ↓ (5.º)

	1.º	2.º	3.º	4.º	5.º
CLAS → 1.º	17	17	16	18	17
2.º	18	18	16	17	16

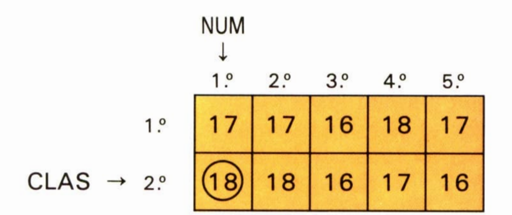
B = ~~0~~ ~~17~~ ~~34~~ ~~50~~ ~~68~~ 85
CLAS = 1
NUM = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6

Como el valor de NUM es mayor de 5, ya se ha terminado la ejecución del bucle interior; entonces se aumenta el contador del bucle exterior.

• *Segunda pasada por el bucle exterior.* CLAS = = ~~1~~ 2 y, como éste no es mayor de 2, se volverá a empezar la ejecución del bucle interior, poniendo la variable NUM otra vez a 1.

NUM = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ 1

• *Primera pasada por el bucle interior.* Nos posicionamos en A(NUM, CLAS) y, como NUM = = 1 y CLAS = 2, en el elemento A(1, 2),



NUM ↓ (1.º)

	1.º	2.º	3.º	4.º	5.º
1.º	17	17	16	18	17
CLAS → 2.º	(18)	18	16	17	16

B = ~~0~~ ~~17~~ ~~34~~ ~~50~~ ~~68~~ ~~85~~ 103
CLAS = ~~1~~ 2
NUM = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~1~~ 2

Como el valor de NUM no es mayor de 5, seguimos dentro del bucle interior.

• *Segunda pasada por el bucle interior*. Nos posicionamos en A(NUM, CLAS) y, como NUM = = 2 y CLAS = 2, en el elemento A(2, 2),

NUM ↓ (2.º)

	1.º	2.º	3.º	4.º	5.º
1.º	17	17	16	18	17
CLAS → 2.º	18	(18)	16	17	16

B = ~~0~~ ~~17~~ ~~34~~ ~~50~~ ~~68~~ ~~85~~ ~~103~~ 121
CLAS = ~~1~~ 2
NUM = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~1~~ ~~2~~ 3

Como el valor de NUM no es mayor de 5, seguimos dentro del bucle interior.

• *Tercera pasada por el bucle interior*. Nos posicionamos en A(NUM, CLAS) y, como NUM = = 3 y CLAS = 2, en el elemento A(3, 2),

NUM ↓ (3.º)

	1.º	2.º	3.º	4.º	5.º
1.º	17	17	16	18	17
CLAS → 2.º	18	18	(16)	17	16

B = ~~0~~ ~~17~~ ~~34~~ ~~50~~ ~~68~~ ~~85~~ ~~103~~ ~~121~~ 137
CLAS = ~~1~~ 2
NUM = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~1~~ ~~2~~ ~~3~~ 4

Como el valor de NUM no es mayor de 5, seguimos dentro del bucle interior.

• *Cuarta pasada por el bucle interior*. Nos posicionamos en A(NUM, CLAS) y, como NUM = 4 y CLAS = 2, en el elemento A(4, 2),

NUM ↓ (4.º)

	1.º	2.º	3.º	4.º	5.º
1.º	17	17	16	18	17
CLAS → 2.º	18	18	16	(17)	16

B = ~~0~~ ~~17~~ ~~34~~ ~~50~~ ~~68~~ ~~85~~ ~~103~~ ~~121~~ ~~137~~ 154
CLAS = ~~1~~ 2
NUM = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

Como el valor de NUM no es mayor de 5, seguimos dentro del bucle interior.

• *Quinta pasada por el bucle interior*. Nos posicionamos en A(NUM, CLAS) y, como NUM = 5 y CLAS = 2, en el elemento A(5, 2),

NUM ↓ (5.º)

	1.º	2.º	3.º	4.º	5.º
1.º	17	17	16	18	17
CLAS → 2.º	18	18	16	17	(16)

B = ~~0~~ ~~17~~ ~~34~~ ~~50~~ ~~68~~ ~~85~~ ~~103~~ ~~121~~ ~~137~~ ~~154~~ 170
CLAS = ~~1~~ 2
NUM = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6

Como el valor de NUM es mayor de 5, salimos del bucle interior y pasamos al exterior, donde se aumentará el valor del contador.

CLAS = ~~1~~ ~~2~~ 3

Como la variable CLAS es mayor de 2, también habrá terminado la ejecución del bucle exterior, con lo cual habrá terminado la ejecución del bucle anidado con las variables y con los valores siguientes:

B = 170 CLAS = 3 y NUM = 6

• *Seguidamente se efectúa el cálculo de la media*. C ← B/10 y, como B = 170, entonces 170/ /10 = 17; por lo tanto C contendrá el valor 17.

• *Impresión de C*. Será la impresión de 17, valor que será la media.

Como puede verse hemos llegado al mismo resultado que cuando lo hemos calculado sin ayuda de la computadora.

Algunos ejemplos de diagramas de flujo

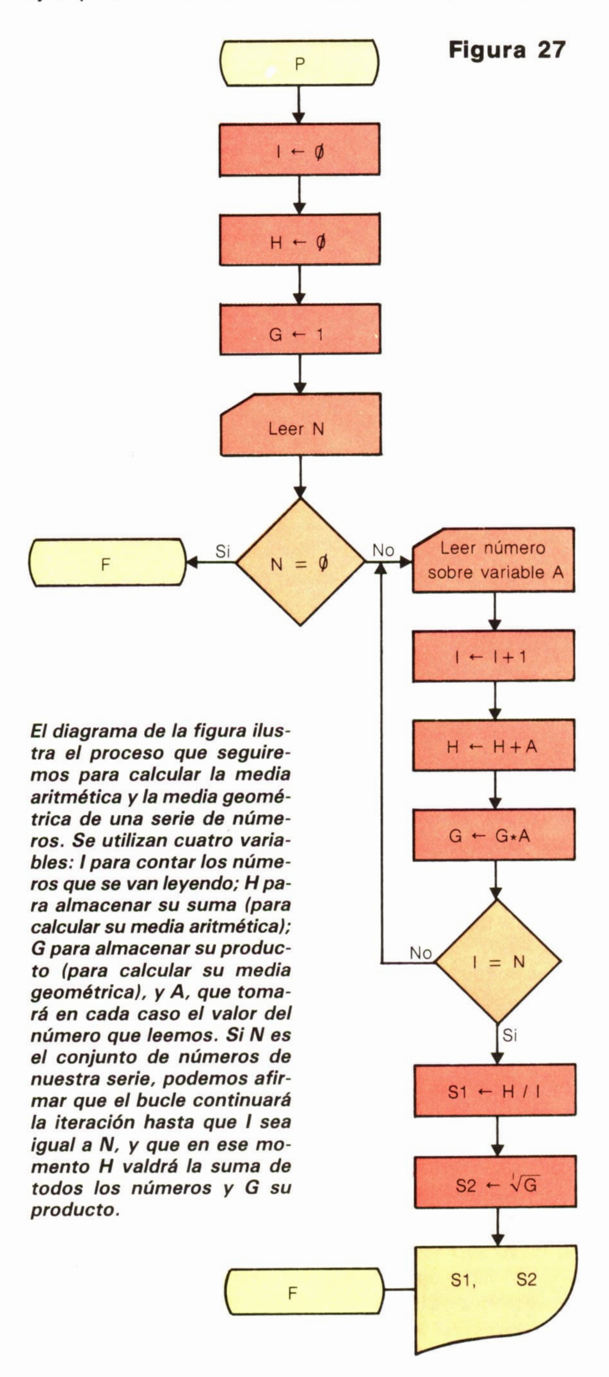
Para terminar este capítulo, vamos a describir unos cuantos ejemplos prácticos de construcción de diagramas; en ellos encontraremos todos los elementos descritos hasta este momento.

EJEMPLO N.º 1

En este ejemplo, dada una serie de números, queremos calcular su media aritmética y su media geométrica. Para realizar estos cálculos construimos el diagrama de la **figura 27**.

Antes de explicar cómo hemos construido este diagrama conviene aclarar cómo se calcula la media aritmética y la media geométrica. La media aritmética de un grupo de números se calcula sumándolos todos y dividiendo el resultado por la cantidad de números sumados. De hecho, ya hemos visto varios ejemplos donde se hacía este cálculo. La media geométrica de un grupo de números se calcula multiplicándolos y sacando la raíz enésima del resultado. Por ejemplo, si tenemos los números 8, 4, 2, los multiplicaremos y al resultado, 64, le sacaremos la raíz cúbica, ya que son 3 números; $\sqrt[3]{64}=4$ porque $4 \times 4 \times 4 = 64$. Por tanto, la media geométrica es 4.

Figura 27



El diagrama de la figura ilustra el proceso que seguiremos para calcular la media aritmética y la media geométrica de una serie de números. Se utilizan cuatro variables: I para contar los números que se van leyendo; H para almacenar su suma (para calcular su media aritmética); G para almacenar su producto (para calcular su media geométrica), y A, que tomará en cada caso el valor del número que leemos. Si N es el conjunto de números de nuestra serie, podemos afirmar que el bucle continuará la iteración hasta que I sea igual a N, y que en ese momento H valdrá la suma de todos los números y G su producto.

En el diagrama vamos a utilizar la variable I como contador de la cantidad de números que vamos a leer; la variable H, para almacenar el resultado de la suma de los números; la variable G, para almacenar el resultado de la multiplicación de los números, y la variable A, en la que se almacenará el número que vamos leyendo.

Empezamos asignando cero a las variables I, H, ya que han de empezar desde cero, y un 1 a la variable G, ya que, si su valor fuera cero, al multiplicar un número por la variable, la primera vez el resultado sería cero y no podríamos seguir. Después leeremos sobre la variable N la cantidad de números que hay que leer. Si N = 0, significa que no vamos a leer ningún número y por lo tanto acabará el proceso, porque no habrá ningún número para realizar los cálculos que queremos hacer.

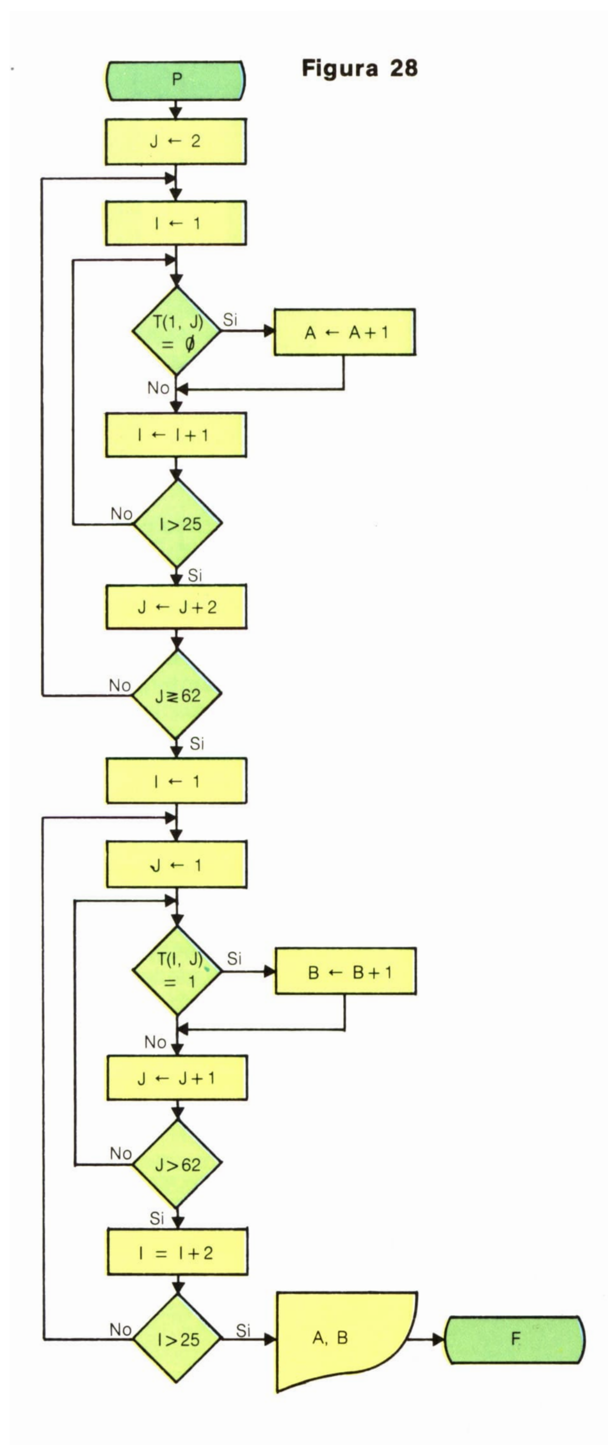
El concepto de lectura está asociado, por ejemplo, a la entrada de números por el teclado. Si N no es igual a cero, leemos o entramos por teclado el primer número que se almacenará en la variable A. Éste es el primer paso de un bucle, dentro del cual vamos sumando los números almacenando siempre el resultado en G. La condición que debe cumplirse para poder salir del bucle es que I (la variable en la que vamos guardando la cantidad de números leídos) sea igual a N (cantidad total de números a leer). Cuando I sea igual a N, en H ya tendremos la suma de todos los números y en G la multiplicación de todos; lo único que queda es dividir la suma por I, con lo que habremos calculado la media aritmética. El resultado lo almacenaremos en la variable S1 y sacaremos la raíz I enésima del resultado de la multiplicación, con lo que habremos calculado la media geométrica, cuyo resultado almacenaremos en S2. Como final, se imprimirán las dos medias S1 y S2.

Para ver el buen funcionamiento del diagrama de la **figura 27**, el lector podría realizar una ejecución paso a paso de éste, tal como se ha realizado anteriormente con otros diagramas de flujo.

EJEMPLO N.º 2

En este ejemplo tenemos una matriz de 25 × 62 elementos, es decir, de 25 columnas por 62 filas: T(25, 62). Cada uno de estos elementos es un número y queremos saber los elementos que son nulos o cero en las filas pares y los elementos que valen 1 en las columnas impares. El diagrama que realiza estos cálculos lo podemos ver en la **figura 28** *(pág. 201)*.

Como vamos a trabajar con una matriz T(25, 62), utilizaremos dos índices. El de las columnas lo tendremos en la variable I y el de



las filas, en la variable J, de tal manera que, al referirnos al elemento T(I, J), nos estamos refiriendo al elemento de la columna I y de la fila J.

Recordemos que en la **figura 17** *(pág. 192)* se explicaba detalladamente cómo localizar un elemento dentro de una matriz habiendo especificado los índices de dicho elemento. Sumaremos 1 a la variable A cada vez que nos encontremos con un elemento nulo en las filas pares y, en cambio, sumaremos 1 a la variable B cuando nos encontremos con elementos que valgan 1 en las columnas impares.

Este proceso, en realidad, se divide en dos: uno para investigar en las filas pares de la matriz y otro para investigar en la columnas impares. Para entender mejor el primer problema, supondremos que la matriz con la que trabajamos no es de 25 × 62, sino que es de 5 × 4, tal como se muestra en la **figura 29**.

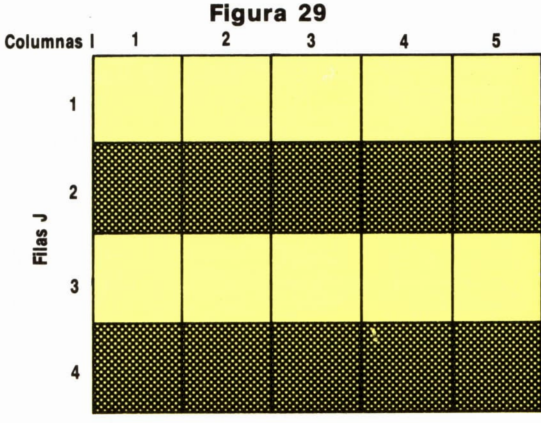
En esta figura, utilizando el índice I para las columnas y el índice J para las filas, vemos que para investigar si hay elementos nulos en las filas pares bastaría solamente comprobar los elementos que están sombreados en la **figura 29**.

Es decir, como J representa las filas (entonces J = 2, primera fila par), se investigarían todos los elementos de la fila 2, que serían T(1, 2), T(2, 2), T(3, 2), T(4, 2), T(5, 2), y después todos los elementos de la fila 4 (última fila par, con J = 4), que serían T(1, 4), T(2, 4), T(3, 4), T(4, 4), T(5, 4). J = 1 y J = 3 no se investigarían, porque no son la filas pares, sino impares. Mientras J toma los valores 2 y 4, I toma todos sus valores 1, 2, 3, 4 y 5. Este sistema se empleará en el problema que tenemos que resolver, con la salvedad de que el número de filas no es 4, sino 62 y el de columnas no es 5, sino 25.

El mismo ejemplo puede servirnos para entender el segundo problema. Supongamos la misma matriz de la **figura 29**; veamos qué elementos deberíamos comprobar para encontrar cuántos elementos de valor 1 hay en las columnas impares. Utilizando otra vez la variable I como índice de las columnas y la variable J como índice de las filas, tendríamos que comprobar los elementos cuyo índice de las columnas fuera impar; por tanto, en este caso, cuando I fuera igual a 1, 3 y 5. Estos elementos serían: con I = 1, T(1, 1), T(1, 2), T(1, 3), T(1,4); con I = 3,

En la matriz de 4 filas y 5 columnas de la figura, si quisiéramos comprobar cuantos elementos nulos figuran en las filas pares deberíamos investigar únicamente el valor de los que aparecen en las cuadrículas sombreadas. Investigando las cuadrículas sombreadas en la figura de la página siguiente, podríamos averiguar por ejemplo, cuantos elementos de valor 1 hay en las columnas impares.

Figura 29



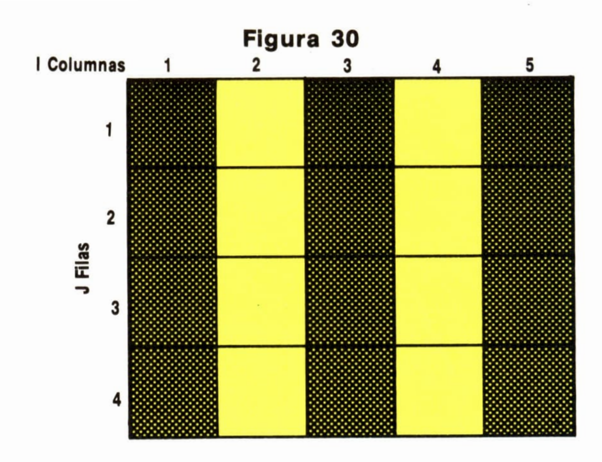
T(3, 1), T(3, 2), T(3, 3), T(3, 4); con I = 5, T(5, 1), T(5, 2), T(5, 3), T(5, 4); que se corresponden con los elementos sombreados de la **figura 30**. Como podemos apreciar, mientras I toma los valores impares 1, 3, 5, J toma todos sus valores 1, 2, 3 y 4. Este mismo sistema se empleará para resolver el problema, con la salvedad de que el número de filas no es 4, sino 62 y el de columnas no es 5, sino 25.

Una vez entendidos mejor los problemas, veamos cómo los resolvemos con el diagrama de la **figura 28**.

Como ya hemos dicho, el proceso, en realidad, se divide en dos; por lo tanto, el diagrama resuelve primero uno y después el otro. Veamos cómo resuelve el primero. Buscaremos, en la matriz de 25 × 62, cuántos elementos de las filas pares son cero. Utilizamos la variable I para indicar las columnas y la variable J para indicar las filas, como siempre.

Empezamos el diagrama asignando el valor 2 a J, pues la investigación se inicia por la primera fila par; también asignamos 1 a J, ya que debemos investigar todas las columnas. Siguiendo estas asignaciones, se investiga si el elemento correspondiente a I, J, T(I,J), tiene valor cero. Si lo tiene, aumentaremos en 1 la variable; en caso contrario, no aumentaremos la variable; en ambos casos aumentaremos en una unidad la variable I, es decir, trasladaremos el índice I a la siguiente fila. Preguntamos si I es mayor que 25, es decir, si se ha llegado a la última columna; si no se ha llegado mediante un bucle, volveremos a investigar el elemento correspondiente, hasta llegar a la última columna, pero siempre conservando la misma fila. Dicho de otro modo, con J = 2 investigamos de I = 1 hasta I = 25. Al llegar a la columna 25, entonces aumentamos la variable J en 2 (porque solamente queremos investigar las filas pares),

Figura 30

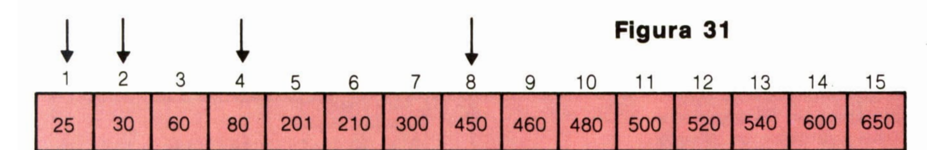


volvemos a realizar el mismo proceso con todas las columnas y así hasta que lleguemos a la última fila, que será la 62, con lo cual en A tendremos el número de elementos de las filas pares que son nulos.

La resolución del primer problema enlaza con la resolución del segundo, que también utiliza la variable I para las columnas y la variable J para las filas. En este caso, se empieza con la asignación de 1 a la variable I, porque empezamos con la primera columna impar; después asignamos 1 a J, porque debemos investigar todas las filas. El proceso es el mismo que el anterior, sólo que, en este caso, mientras I vale 1, J va variando de 1 a 62: mientras J no sea mayor de 62, si se encuentra algún elemento que sea 1, se aumentará en una unidad la variable B; cuando J sea mayor de 62, se incrementará I en 2 para seleccionar la siguiente columna impar, y así sucesivamente hasta que I sea mayor de 25, lo cual querrá decir que hemos llegado al final y tendremos en B el número de elementos que hemos encontrado cuyo valor es



Actualmente son muchas las empresas que emplean computadoras para llevar la contabilidad. Existe en el mercado un software que imita las hojas de contabilidad tradicionales; permite disponer de filas y columnas y entrar cifras en las cuadrículas. Después puede calcular cosas tales como el porcenaje que habrá que pagar de impuestos, la comisión que deberá darse a los vendedores y los gastos generales, y proporcionar una salida impresa a pie de página del beneficio correspondiente. Permite además experimentar con las condiciones futuras del negocio: puede, por ejemplo, variarse una de las cifras que determinan los costes y el software calculará todas las demás.



La figura muestra un vector de 15 elementos, que ilustra el juego de adivinar el lugar en que se encuentran 15 personas, cada una de las cuales tiene asignado un número distinto.

la unidad. Finalizamos imprimiendo los valores de las variables A y B.

Si nos fijamos, en cada uno de los casos estamos utilizando un bucle anidado para resolver el problema. Estos bucles siguen la misma técnica que se ha utilizado en el apartado «Bucles anidados» *(pág. 196)*, donde precisamente se explica la manera de trabajar con matrices mediante dichos bucles.

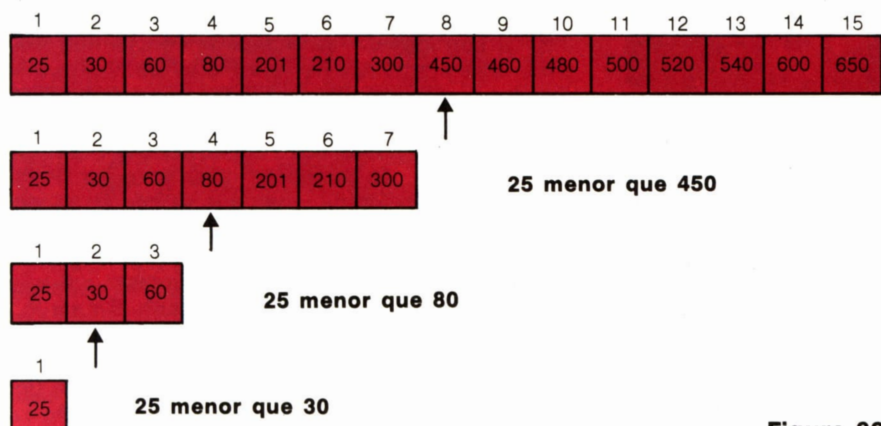
EJEMPLO N.º 3

En este ejemplo vamos a ver la solución a un juego que el lector puede realizar con sus amigos; es un juego puramente matemático. Se trata de lo siguiente: dado un número de personas y teniendo cada una de ellas un número asignado (números que pueden estar, por ejemplo, entre 1 y 1.000), al colocar a las personas en orden decreciente o creciente de números, tratar de adivinar qué persona tiene un número determinado, con el menor número de preguntas posibles. El método que vamos a explicar es eficiente en el caso de que el número de personas sea potencia de dos menos 1, es decir, 1, 3, 7, 15, 31, 63, 127, 255, 511, ...

En tal caso, el número de personas coincidirá con uno de estos números, con lo que se podrá decir, sin temor a equivocarse, que el número de preguntas que hay que hacer es, como máximo, el de la potencia, es decir: si tenemos 1 persona, como $1=2^1-1$, haciendo una pregunta sabremos dónde se encuentra el número; con $3=2^2-1$, dos preguntas, con $7=2^3-1$, tres preguntas, con $15=2^4-1$, cuatro preguntas y así sucesivamente. Podemos decir, pues, que con N personas, siendo $N=2^K-1$, se necesitarán K preguntas.

Para resolver el problema mediante un diagrama, supongamos que cada persona es un elemento de un vector y que el número de personas es 15, con lo que tendremos un vector de 15 elementos. Cada elemento, como ya hemos dicho, tendrá un valor y estos valores pueden ir de 1 a 1.000. Lo elementos estarán clasificados de menor a mayor. Para ilustrar este ejemplo se ha construido el vector de la **figura 31**.

El método se basa en ir dividiendo la tabla en mitades, recibiendo el nombre de búsqueda dicotómica. Por ejemplo, en el caso de la **figura 31**, intentaremos buscar el número 25. Lógicamente, la persona que busca el número no sabe qué números hay en cada uno de los elementos; por lo tanto, el número 25 tanto puede encontrarse en el elemento 1 como en el elemento 15. Lo que sí podemos asegurar es que con 4 preguntas como máximo sabremos dónde se encuentra, siempre que ese número exista en algún elemento del vector. Un aspecto que debemos hacer notar es que, si los números no están clasificados de menor a mayor o de mayor a menor, este método no sirve absolutamente para nada. Como decíamos, el método se basa en dividir la tabla en mitades. Por ejemplo, en este caso nos vamos al elemento 8 y ahí preguntamos si el valor del elemento es $<$, $>$ o $=$ al número buscado. Como 25 es menor de 450, nos contestarían que es menor. Entonces, como estamos en el elemento 8, nos trasladamos al elemento 4, y ahí volvemos a hacer la pregunta: si el valor del elemento es $<$, $>$ o $=$ al número buscado; en este caso será $<$. Entonces nos vamos al elemento 2 y volvemos a preguntar $<$, $>$ o $=$ al elemento buscado. Como es menor, solamente puede ser el elemento



Cuando el número de personas con que se realiza el juego es una potencia de 2 menos 1 (en nuestro ejemplo $15=2^4-1$), podremos averiguar donde se encuentra cada número realizando como máximo un número de preguntas igual al de la potencia de 2 (en nuestro ejemplo 4). El método que seguimos se denomina de búsqueda dicotómica y consiste en ir dividiendo el vector en mitades, preguntando en cada caso si el número que buscamos se encuentra a la derecha o a la izquierda del número central.

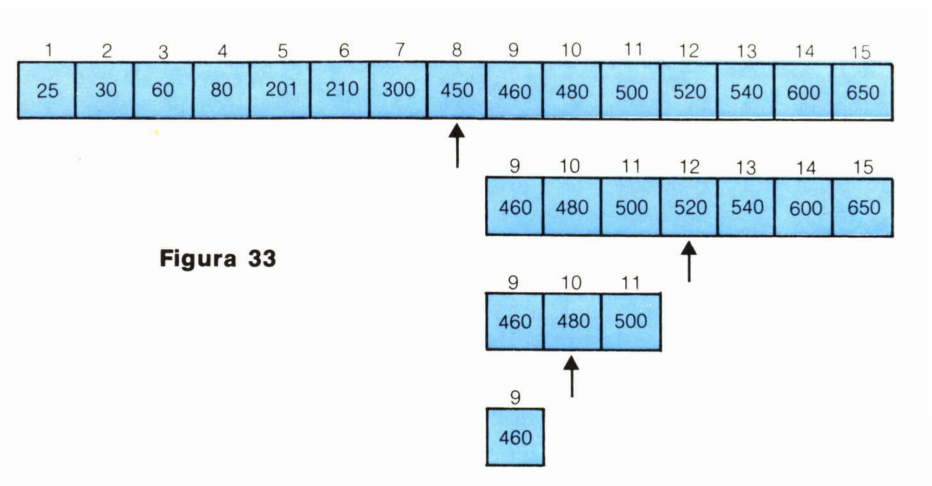


Figura 33

La figura ilustra como buscaríamos, en nuestro juego, el número 460. Empezaríamos preguntando en la casilla 8 si el valor de esa casilla es mayor o menor que el del número que buscamos. Nos contestarían que menor, por lo que proseguiríamos investigando a la derecha de esa casilla. Preguntaríamos lo mismo en la casilla 12, la respuesta sería que es menor, con lo que sabríamos que el número que buscamos se encuentra en una de las casillas entre la 8 y la 12. Preguntaríamos en la casilla 10, y si nos contestan que es menor, sabemos que se encuentra en la 9.

número 1, como efectivamente es. A pesar de ello, nos desplazaremos a este elemento y haremos la pregunta. Si nos responden >, nos están engañando; si nos responden <, o nos están engañando o el número no existe. Solamente nos pueden contestar que es igual.

Siempre que nos contesten «mayor», los elementos que hay que considerar quedarán limitados entre el elemento que investigamos y el final de la tabla hacia la derecha; siempre que nos contesten menor, será al revés, hacia la izquierda. En el caso explicado, se ha dividido la tabla tal como se muestra en la **figura 32**.

Otro ejemplo diferente sería buscando el número 460 como en la **figura 33**.

El número 460 lo encontramos en el elemento 9. Tanto en el primer caso como en el segundo, hemos encontrado los elementos con 4 pasos. Sin embargo, si buscáramos, por ejemplo, el número 520, con 2 preguntas habría sido suficiente. Veamos ahora cómo hemos plasmado este método en un diagrama fijándonos en la **figura 34** *(pág. 205)*.

Como primer paso, leeremos el número que queremos buscar y el número de elementos del vector cuyo nombre es A. Estos dos números los asignaremos a las variables X y N. Utilizaremos la variable F para ir sumando el número de preguntas que vamos haciendo. Por lo tanto, le asignaremos como primer paso un 1, porque como mínimo haremos una pregunta.



Vamos a utilizar la variable I para que nos indique siempre el elemento máximo al que podremos acceder y la variable H para indicar el elemento mínimo al que también podremos acceder. En este caso, al empezar el diagrama, como tenemos un vector de 15 elementos, asignaremos el valor de N a la variable I y el valor de I a la variable H; por lo tanto H = 1 y N = 15, que son los límites del vector. Después utilizaremos la variable J para ir buscando el elemento central después de cada comparación. Para ello, la primera vez dividiremos el número de elementos de la tabla N por 2 y, como la división tiene que ser entera, es decir, sin decimales, al resultado le sumaremos 1. Vamos a utilizar también la variable K como índice que nos apuntará al elemento del vector al que estamos accediendo cada vez para compararlo con el número buscado. Siguiendo el método descrito anteriormente, igualamos K con J la primera vez, porque el elemento que hay que comparar es el central del vector.

El siguiente paso es la comparación del número buscado con el valor del elemento A(K), es decir, el elemento del vector A al cual apunta K. Si son iguales, ya hemos encontrado el número de preguntas hechas P y habremos terminado. Si el número buscado es menor que el del elemento, acotamos el número de elementos que vamos a comparar por la derecha, de tal manera que nos quedarán solamente los de la mitad de la izquierda, cosa que se consigue con I ← I−J; volvemos a buscar el elemento central del vector que nos queda con J ← J / 2 y K ← K−J. Si el número es mayor, acotamos el número de elementos a comparar por la izquierda, mediante la variable H, con H ← H+J y volvemos a buscar el elemento central del valor acotado con J ← J / 2 y K ← K+J. Tanto en un caso como en el otro, preguntamos a continuación si J = 0. Si la respuesta es sí, quiere decir

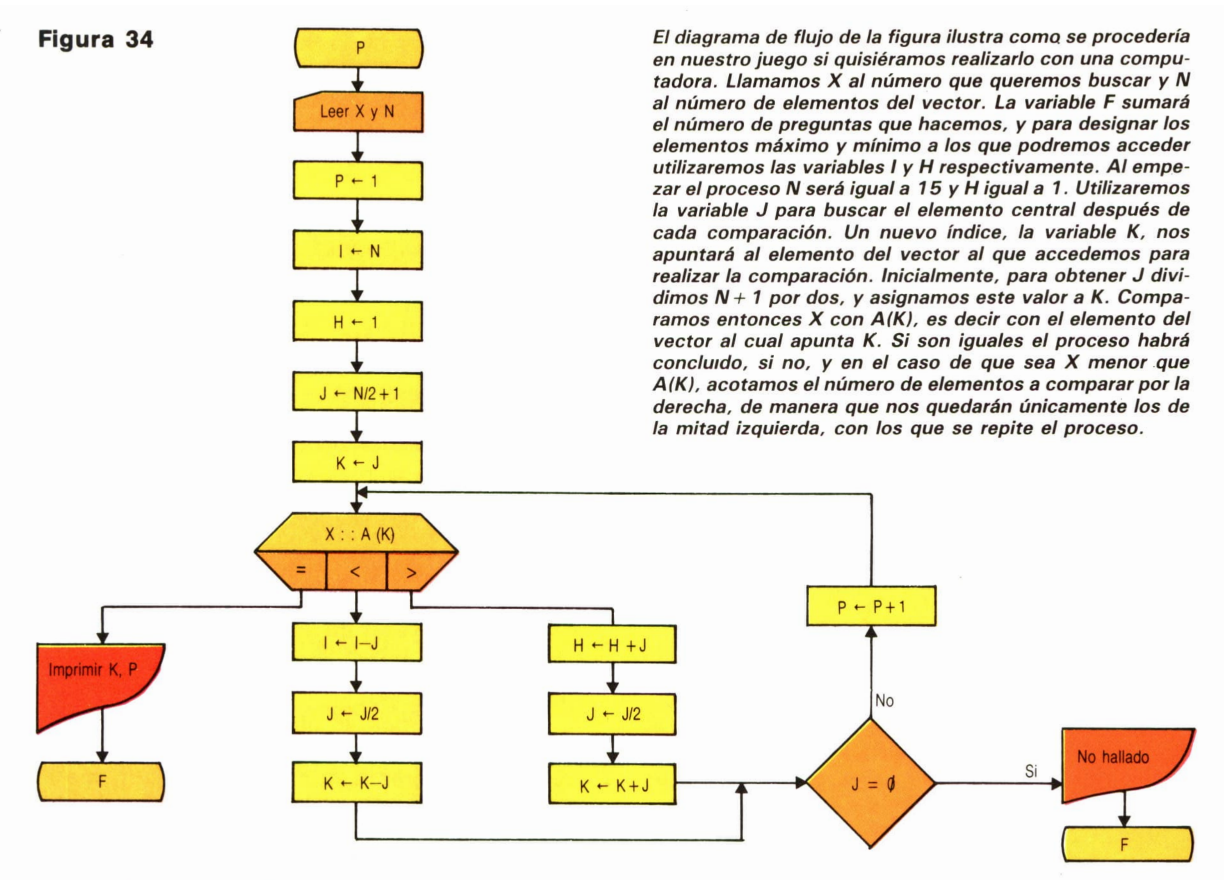


Figura 34

El diagrama de flujo de la figura ilustra como se procedería en nuestro juego si quisiéramos realizarlo con una computadora. Llamamos X al número que queremos buscar y N al número de elementos del vector. La variable F sumará el número de preguntas que hacemos, y para designar los elementos máximo y mínimo a los que podremos acceder utilizaremos las variables I y H respectivamente. Al empezar el proceso N será igual a 15 y H igual a 1. Utilizaremos la variable J para buscar el elemento central después de cada comparación. Un nuevo índice, la variable K, nos apuntará al elemento del vector al que accedemos para realizar la comparación. Inicialmente, para obtener J dividimos N + 1 por dos, y asignamos este valor a K. Comparamos entonces X con A(K), es decir con el elemento del vector al cual apunta K. Si son iguales el proceso habrá concluido, si no, y en el caso de que sea X menor que A(K), acotamos el número de elementos a comparar por la derecha, de manera que nos quedarán únicamente los de la mitad izquierda, con los que se repite el proceso.

que el número no existe o que nos están engañando. En caso negativo, aumentamos en 1 el valor de F y volvemos a comparar y así sucesivamente hasta encontrar el número.

No olvidemos que todas las divisiones efectuadas en este diagrama son enteras, o sea, sin decimales; lo cual quiere decir que el resultado de dividir 1 por 2 no es 0,5, sino 0.

Este diagrama tiene más dificultad de comprensión que los descritos anteriormente, porque utiliza muchas variables como punteros. Por lo tanto, para que quede todo más claro, vamos a realizar el seguimiento de la ejecución del diagrama paso a paso en un par de casos.

Supongamos el vector de la **figura 31** y que el número que buscamos es el 480. Siguiendo paso a paso la ejecución, tendremos:

- Asignaciones de las variables antes de realizar la primera comparación:

X = 480   N = 15   P = 1   I = 15   H = 1

Como J ← N/2 + 1, J ← 15/2 + 1, J ← 7 + + 1, J = 8 y K = 8, los punteros quedarán colocados como en la **figura 35**. H en el primer elemento, I en el último y K en el elemento central.

Como K siempre nos apunta al elemento que vamos a comparar, después de la primera comparación las variables nos quedarán como sigue.

Como la comparación de 480 con 450 saldrá por mayor, entonces

X = 480   N = 15   P = 1
I = 15   H = ~~1~~ 9   J = ~~8~~ 4   K = ~~8~~ 12

con lo que se habrán modificado los valores de H, J y K, ya que las asignaciones correspondientes a la salida por mayor son H ← H + J, J ← J / 2 Y K ← K + J.

Por tanto el vector quedará con los punteros tal como en la **figura 36**.

Una vez efectuadas las asignaciones, hemos acotado el vector a los elementos 9 al 15, porque del 1 al 8 ya sabemos que son menores de 450.

El siguiente paso es preguntar si J = 0; al ser la respuesta negativa, aumentamos P, porque vamos a hacer otra comparación

P = ~~1~~ 2

- *Siguiente comparación*. Como K apunta al elemento 12, compararemos 480 con 520; saldrá por menor y, después de las asignaciones correspondientes, las variables quedarán de este modo:

X = 480   N = 15   P = 2
I = ~~15~~ 11   H = ~~1~~ 9   J = ~~8~~ ~~4~~ 2
K = ~~8~~ ~~12~~ 10

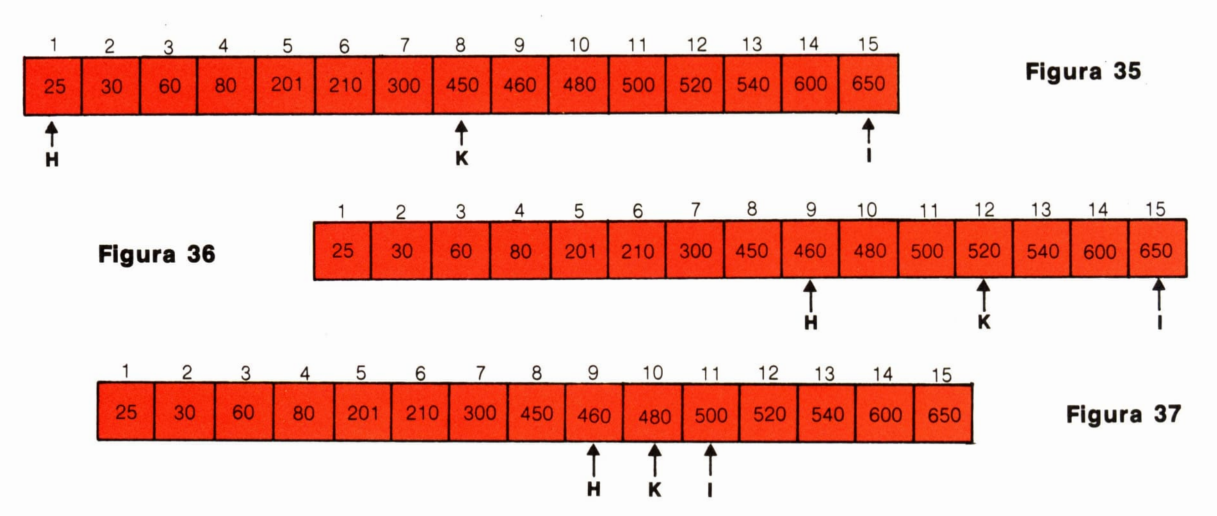


Figura 35

Figura 36

Figura 37

Una vez efectuadas las asignaciones, hemos acotado el vector a los elementos del 9 al 11. El siguiente paso es preguntar si J = 0; como no lo es, aumentamos P, porque vamos a hacer otra comparación.

P = ~~1~~ ~~2~~ 3

• *Siguiente comparación*. Como K apunta al elemento 10, compararemos 480 con 480. Saldrá por igual, con lo que se imprimirá K y P; es decir, el valor 480 está en el elemento 10 y con 3 comparaciones lo hemos encontrado, porque K = 10 y P = 3.

El diagrama construido para resolver este problema puede modificarse y reducirse para que no se tengan que usar tantas variables.

En la **figura 38** tenemos el mismo diagrama, pero más sencillo.

Si nos fijamos, lo único que hemos omitido han sido las asignaciones de las variables I y H, que en el diagrama anterior solamente nos eran útiles para indicarnos qué elementos estábamos

Figura 38

P
Leer X, N
P ← 1
J ← N/2 + 1
K ← J
X : : A (K)
= < >
Imprimir
F
J ← J/2
K ← K–J
J ← J/2
K ← K+J
J = ∅
No
P ← P+1
Si
No hallado
F

El diagrama de flujo que hemos construido para nuestro juego (fig. de la página anterior) utiliza un gran número de variables y es posible simplificarlo un poco para no tener que utilizar tantas. De hecho, las variables I y H no tenían más función que designar los elementos máximo y mínimo a los que podíamos acceder, es decir acotar el vector que estabamos investigando, pero no son imprescindibles para la realización del proceso. Si prescindimos de ellas, como hemos hecho en el diagrama de flujo adjunto, podremos incluir dos órdenes de ejecución menos, correspondientes a los dos caminos, por la derecha y por la izquierda, por los que puede proseguirse nuestra investigación a partir de la comparación con un elemento central.

acotando en el vector; pero, en realidad, no era necesario conocer estos datos.

Si el lector quiere seguir paso a paso la ejecución de este diagrama, verá que llega a los mismos resultados que con el anterior. He aquí un ejemplo de dos diagramas diferentes para la resolución de un mismo problema.

Necesidad de los diagramas de flujo

Los diagramas son necesarios para llegar a analizar y resolver un problema en toda su profundidad, cosa que realmente es necesaria para poder programar computadoras, ya que, como hemos dicho en otras ocasiones, la computadora no hará nada que previamente no se le haya ordenado.

En todas las estructuras y ejemplos que hemos visto se ha podido comprobar cómo se analizaba un determinado proceso. Se trataba de la búsqueda de un método para resolver el problema y luego, a partir de este método y mediante algunas estructuras como las descritas, de construir un diagrama en el que se plasmara el método con una terminología determinada.

Todo este proceso es conveniente hacerlo siempre que se quiera programar una computadora, por sencillo que sea el problema que deseemos solucionar con ella.

Consideramos muy importante, incluso antes de aprender cualquier lenguaje de programación, que se sepan construir diagramas; esto significa que, cuando se aprenda dicho lenguaje y se construya con él un programa, se cometerán errores propios del desconocimiento del lenguaje de programación, pero no se cometerán errores por desconocimiento del método de resolución del programa, ya que antes se habrá construido el diagrama que resolverá el problema mediante el método apropiado.

Si no se hace así, un principiante que quiera construir un programa tendrá muchos más problemas de los habituales.

SISTEMAS DE NUMERACIÓN

Las computadoras trabajan con el sistema de numeración que se llama binario, por lo tanto es muy conveniente saber en qué consiste y cómo pasar de éste a otros sistemas de numeración (decimal, octal y hexadecimal), ya que es uno de los conceptos que vamos a encontrar con más frecuencia cuando construyamos programas.

El sistema de numeración más comúnmente conocido y utilizado es el decimal, que utiliza 10 símbolos:

Ø, 1, 2, 3, 4, 5, 6, 7, 8 y 9

El sistema binario utiliza dos símbolos:
Ø y 1

El sistema octal utiliza 8 símbolos:
Ø, 1, 2, 3, 4, 5, 6, y 7

El sistema hexadecimal utiliza 16 símbolos:
Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9,
A, B, C, D, E y F, siendo el valor de

A = 10
B = 11
C = 12
D = 13
E = 14
F = 15

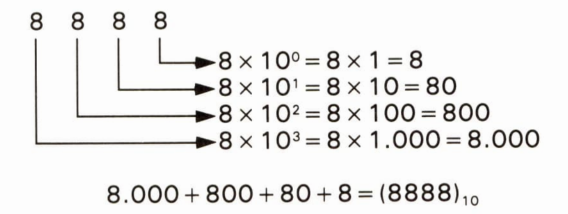
En este último sistema (y en todos los demás), para representar valores no expresados directamente por sus símbolos, se usan agrupaciones de tales símbolos. Así cada símbolo o cifra tiene un valor diferente, aparte del propio, según la posición que ocupa en la agrupación.

Este valor está en función de la base, que es igual al número de símbolos con que cuenta el sistema. De esta manera, el sistema decimal, al utilizar 10 símbolos, es de base 10; el binario, al utilizar 2, es de base 2; el octal, al utilizar 8, es de base 8, y el hexadecimal, por tanto, al utilizar 16, es de base 16.

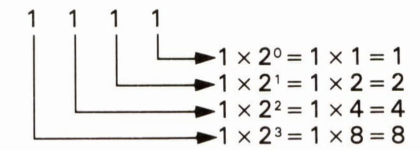
VALOR DEL SÍMBOLO * VALOR DE LA POSICIÓN

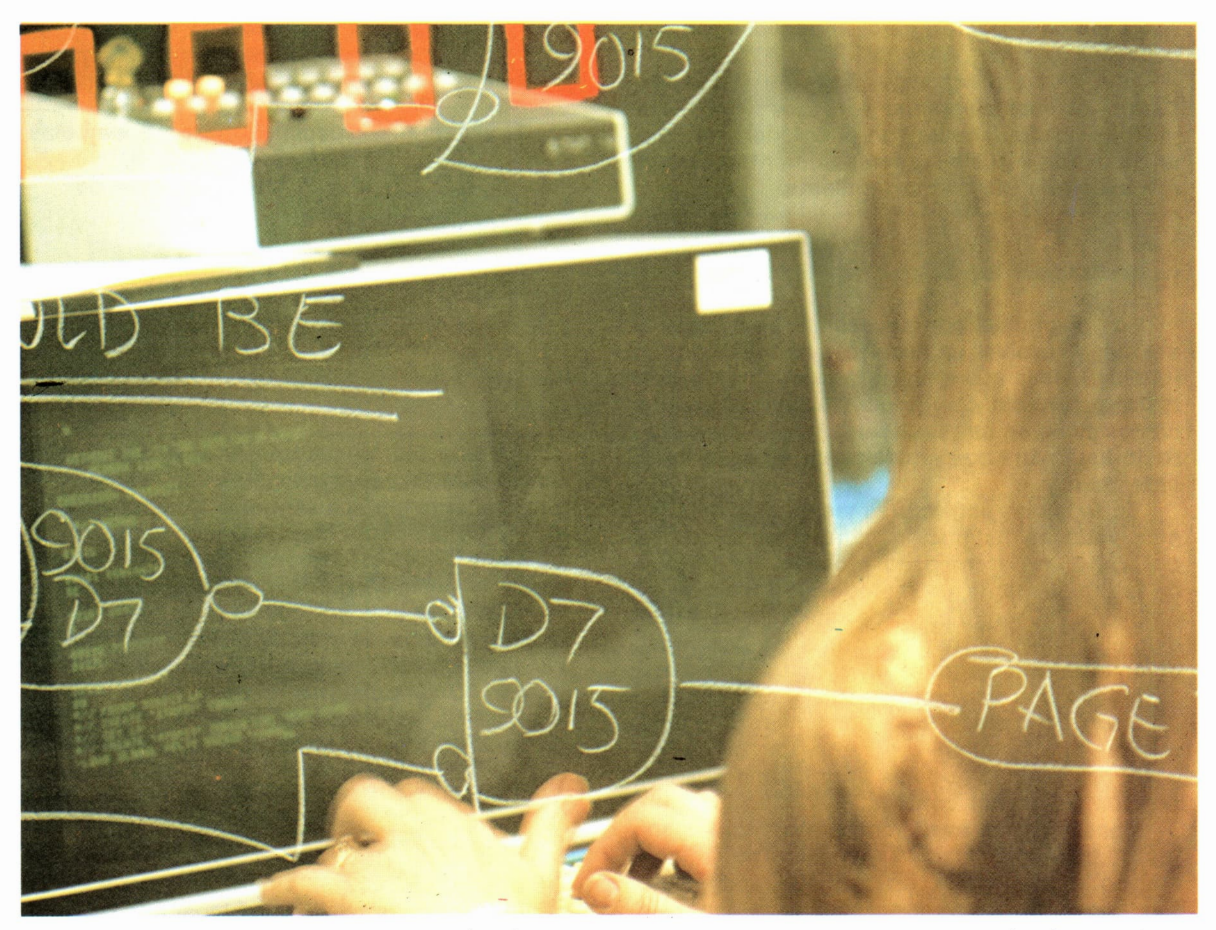
El valor de la posición es igual a la base del sistema elevada a potencias progesivas desde «Ø» en adelante, contando a partir de la posición menos significativa, es decir, desde la cifra colocada más a la derecha.

Por ejemplo, en el valor decimal 8888, vemos que el mismo símbolo o cifra toma valores diferentes, porque está colocada en posiciones diferentes. Así:



Por ejemplo, en el valor binario 1111, sucede lo mismo. Así:





Cuando un usuario utiliza una computadora hay una situación subyacente que en este montaje fotográfico se ha hecho transparente : la de la existencia de unas estructuras lógicas comunes al usuario y a la máquina.

Como podemos ver, se realiza el mismo cálculo que en base 10, pero la base en este caso es 2. También estamos empezando a ver cómo se realiza la transformación de un número binario en decimal, pero esto lo explicaremos con más detalle más adelante.

Haciendo lo mismo con el valor octal 555,

5 5 5

$5 \times 8^0 = 5 \times 1 = 5$
$5 \times 8^1 = 5 \times 8 = 40$
$5 \times 8^2 = 5 \times 64 = 320$

Y por último realizándolo con el valor hexadecimal AAA, vemos también que cifras iguales tienen diferente valor si están colocadas en posición diferente.

A A A

$A \times 16^0 = 10 \times 1 = 10$
$A \times 16^1 = 10 \times 16 = 160$
$A \times 16^2 = 10 \times 256 = 2.560$

A partir de este punto nos centraremos sobre los tres principales sistemas de numeración: el binario, el hexadecimal y el decimal; aunque éste no lo vamos a explicar, porque es conocido por todos. Dejaremos aparte el sistema octal, por ser el menos utilizado.

Números binarios

Las cifras del sistema binario en el campo de la computación se llaman bits. Como ya se ha indicado, un número binario sólo puede tener las cifras 0 o 1, por lo que denominaremos bit tanto a la cifra cero como a la cifra 1.

Normalmente, cuando nos encontramos con un valor en binario, nunca pensamos realmente en dicho valor como tal, sino que siempre lo transformamos en un valor en base 10 o decimal, porque éste es el sistema de numeración con el que habitualmente pensamos y calculamos. Sin embargo, alguien que quiera familiarizarse a fondo con el mundo de la computación deberá acostumbrarse a trabajar con valores binarios sin transformarlos en decimales.

Paso de binario a decimal

De todas formas, veamos cómo se pasa de un valor binario a su correspondiente decimal.

Supongamos el valor binario 1010. Vamos a transformarlo en decimal. Empezaremos por calcular el valor de cada una de las cifras del número, tal y como hemos visto, según la fórmula descrita.

De esta manera:

— calculando el valor de la cifra menos significativa, es decir, el primer cero de la derecha, tendremos que su valor será 0×2^0, cuyo resultado es 0×1, que es 0;

— calculando el valor de la segunda cifra, obtendremos:

$$1 \times 2^1 = 41 \times 2 = 2;$$

— calculando el valor de la tercera cifra, obtendremos:

$$0 \times 2^2 = 0 \times 4 = 0;$$

— calculando el valor de la cuarta cifra obtendremos:

$$1 \times 2^3 = 1 \times 8 = 8$$

Una vez calculados los valores de las cifras, simplemente los sumaremos entre sí para calcular el valor decimal correspondiente al valor binario 1010.

Así: $0+2+0+8 = 10$, con lo que podemos decir que $(1010)_2 = (10)_{10}$, es decir, que 1010 en base 2 es igual que 10 en base 10.

Concepto de byte

Como vemos, utilizamos configuraciones de bits (0 y 1) para representar valores decimales; esto es lo mismo que realiza la computadora, pero ella también puede trabajar con caracteres y, para hacerlo, también utiliza configuraciones binarias. Al espacio que ocupa un carácter dentro de la memoria de la computadora se le denomina byte, que no es más que una agrupación de 8 bits.

Así por ejemplo, el valor binario 11111111 es el mayor valor que puede representarse en un byte (8 bits) y su valor decimal es:

$$1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 + 1 \times 2^6 + 1 \times 2^7 = 1+2+4+8+16+32+64+128 = 255$$

Por lo tanto, el valor $(11111111)_2 = (255)_{10}$ y este valor 255 será el valor decimal mayor que puede representarse dentro de un byte.

Antes, al hablar de caracteres, incluíamos dentro de este término a las letras. Por tanto, podemos decir que en un byte representamos una letra; pero tal representación no se hace mediante un dibujo de la letra dentro del byte, sino que todas las computadoras, en su sistema operativo, poseen una tabla que les informa de las correspondencias entre cada carácter y los valores de 0 a 255 (valor mínimo y máximo que pueden almacenarse en un byte). Así por ejemplo, en una de estas tablas de caracteres ASCII, el valor correspondiente a la letra *A* (mayúscula) es 65 decimal y el correspondiente a la letra *a* (minúscula) es 97 decimal. (Véase la **tabla de caracteres ASCII** que aparece en el volumen I, *página 162*.)

Así, la representación binaria de la letra *A* mayúscula dentro de un byte en la computadora sería «01000001», porque $1 \times 2^0 + \emptyset \times 2^1 + \emptyset \times 2^2 + \emptyset \times 2^3 + \emptyset \times 2^4 + \emptyset \times 2^5 + \emptyset \times 2^6 + \emptyset \times 2^7 = 1 + 64 = 65$ decimal.

El correspondiente a la letra *a* minúscula será «01100001», porque $1 \times 2^0 + \emptyset \times 2^1 + \emptyset \times 2^2 + \emptyset \times 2^3 + \emptyset \times 2^4 + \emptyset \times 2^5 + \emptyset \times 2^6 + \emptyset \times 2^7 = 1 + 32 + 64 = 97$ decimal.

Por lo tanto, cuando la computadora reconozca estos valores dentro de un área alfanumérica de datos, los tratará como *A* y *a*; es decir, si debe imprimir estos bytes en pantalla, imprimirá una *A* y una *a* y no un 65 y un 97, aunque internamente los tenga con este valor.

Por ejemplo, cuando estamos entrando datos a la computadora y le digitamos la palabra LUNES, almacenará esta palabra (si se utiliza la tabla ASCII) con los valores 76, 85, 78, 69 y 83 y, más exactamente, como:

L	U	N	E	S
01001100	01010101	01001110	01000101	01010011

Paso de decimal a binario

Hemos visto cómo transformamos los valores binarios en decimales, pero es necesario también saber cómo se realiza el proceso contrario, que se efectúa de la siguiente manera.

Se toma el número y se divide por 2, el resultado de la división se vuelve a dividir por 2 y así sucesivamente hasta llegar a un resultado menor de 2. Cuando se haya producido esto, se irán escribiendo, de izquierda a derecha, primero el cociente de la última división y después el resto de cada una de las divisiones anteriores, de más reciente a más antigua. Por ejemplo, para pasar a binario el valor 132, haríamos:

1.ª: 132 | 2 → 12, 66, resto 0

2.ª: 66 | 2 → 06, 33, resto 0

3.ª: 33 | 2 → 13, 16, resto 1

4.ª: 16 | 2 → 8, resto 0

5.ª: 8 | 2 → 4, resto 0

6.ª: 4 | 2 → 2, resto 0

7.º: 2 | 2 → 1, resto 0

Entonces 1 será el cociente de la última división.

Si seguimos examinando las operaciones realizadas en el paso de decimal a binario:

Ø será el resto de la última división
Ø será el resto de la 6.ª
Ø será el resto de la 5.ª
Ø será el resto de la 4.ª
1 será el resto de la 3.ª
Ø será el resto de la 2.ª
Ø será el cociente de la 1.ª última división
El número binario será 1 Ø Ø Ø Ø 1 Ø Ø.

Pongamos otro ejemplo para ir comprendiéndolo mejor. Supongamos el número decimal 63. Entonces haremos como antes:

```
 1.ª        2.ª        3.ª       4.ª      5.ª
63 |2      31 |2      15 |2     7 |2     3 |2
03  31     11  15      1  7     1  3     1  1
 1          1
```

El valor binario será, por lo tanto, 111111.

Números hexadecimales

Con los números hexadecimales ocurre lo mismo que con los números binarios; es decir, nunca pensamos en dicho valor como tal, sino que siempre se transforma en un valor en base 10 o decimal.

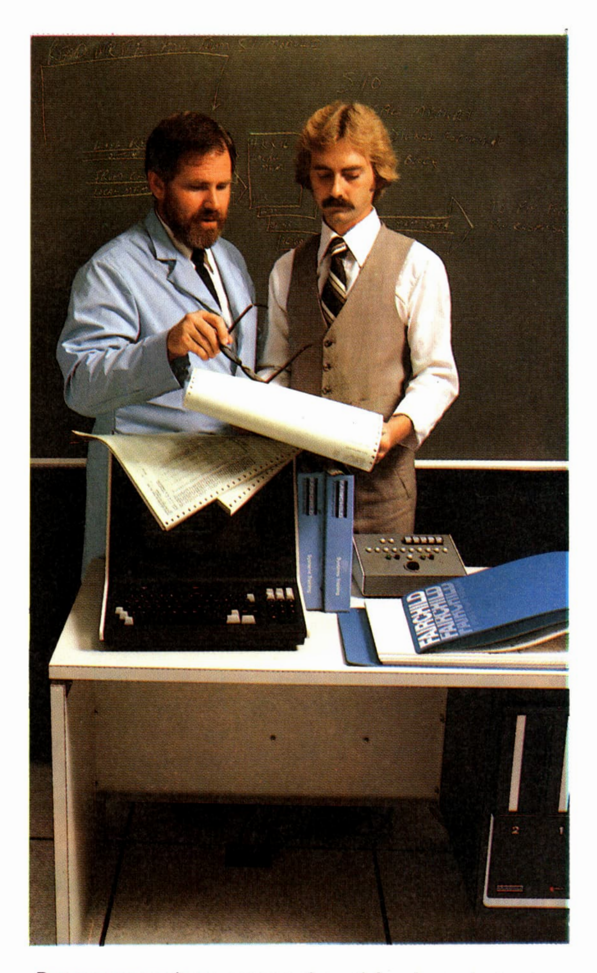
Al trabajar con computadoras nos encontraremos muchas veces con que la máquina nos da muchos datos con valores hexadecimales. Así, si le pedimos a una computadora que nos imprima en impresora o nos visualice en la pantalla una determinada zona de su memoria, los valores que nos dará estarán expresados en hexadecimal. También ocurrirá lo mismo con listados de programas compilados, en los que las direcciones de cada instrucción e incluso los datos numéricos aparecerán en hexadecimal.

Una de las razones por las cuales se produce esto es que resulta mucho más fácil la transformación de datos binarios a hexadecimales que a decimales.

Paso de hexadecimal a decimal

Para realizar la transformación de un número hexadecimal a uno decimal, se utiliza el mismo método que el descrito en el paso de binario a decimal, pero cambiando la base de la potencia, que en lugar de ser 2 será 16.

Así, por ejemplo, con el número hexadecimal 5AC haríamos lo siguiente: calcularíamos primero el valor de la cifra menos significativa, que es la C, de tal manera que multiplicaríamos su valor por la potencia de base 16 y exponente cero, ya que es la 1.ª cifra, $C \times 16^0$. Como ya hemos explicado antes, el valor de C es 12 (recordemos que A = 10, B = 11, C = = 12, D = 13, E = 14 y F = 15 en el sistema de numeración hexadecimal); entonces obtendríamos 12×16^0, 16^0 es igual a 1 y el resultado, por tanto, será $12 \times 1 = 12$.



Dos programadores comprueban el funcionamiento de un nuevo software.

Calculando el valor de la 2.ª cifra obtendríamos $A \times 16^1 = 10 \times 16 = 160$ y, por último, calculando el valor de la última cifra obtendríamos $5 \times 16^2 = 5 \times 256 = 1.280$.

Obtenidos los valores de cada cifra, solamente nos quedará sumarlos todos y obtendremos el valor decimal correspondiente:

$$1.280 + 160 + 12 = 1.452$$

Por lo tanto, podremos afirmar que $(5AC)_{16} = (1.452)_{10}$.

Otro ejemplo: el valor hexadecimal 72B.

7 2 B

- B → $B \times 16^0 = 11 \times 16^0 = 11 \times 1 =$ 11
- 2 → $2 \times 16^1 = 2 \times 16 =$ 32
- 7 → $7 \times 16^2 = 7 \times 256 =$ 1.792

Sumando, $11 + 32 + 1.792 = 1.835$

Entonces diremos que $(72B)_{16} = (1.835)_{10}$.

Paso de decimal a hexadecimal

Es conveniente saber también cómo se efectúa la conversión de un valor decimal en hexadecimal, es decir, el proceso contrario al explicado anteriormente.

Tomemos un valor decimal, por ejemplo, 1.835, que antes hemos visto que se correspondía con el valor hexadecimal 72B, y veamos cómo hacemos la transformación.

Dividimos el valor decimal por 16, el cociente resultante lo volvemos a dividir por 16 y así sucesivamente hasta llegar a un cociente que sea menor de 16. De esta manera:

```
        1.ª                2.ª
1835 | 16            114 | 16
023    114           02    7
 075
  11
```

Una vez hechas las divisiones, para encontrar el valor decimal, se tomará primero el cociente de la última división, que será la cifra más significativa del valor hexadecimal, y después el resto de esta misma división como la siguiente cifra significativa, el resto de la siguiente división como siguiente cifra y así sucesivamente hasta el resto de la primera división.

De esta manera obtendremos 7, 2 y 11 y como la cifra 11 en hexadecimal es la letra B, el valor será 72B, que es el resultado al que se esperaba llegar.

Supongamos el otro ejemplo anterior, es decir, el valor decimal 1.452; haciendo la misma transformación

```
        1.ª             2.ª
1452 | 16            90 | 16
 012   90            10   5
```

obtendremos las cifras 5, 10 y 12. Como la cifra 10 en hexadecimal es la letra A y la cifra 12 la letra C, obtendremos como valor hexadecimal 5AC, que es el resultado correcto.

Paso de binario a hexadecimal y de hexadecimal a binario

Muchas veces necesitaremos convertir directamente valores binarios a hexadecimales, o al revés; es decir, conversiones directas, sin necesidad de tener que pasar primero a decimal y después tener que hacerlo a binario o hexadecimal.

Estas conversiones son factibles y más rápidas que las que hemos visto hasta ahora.

El primer caso, es decir, la conversión de binario a hexadecimal, se efectúa del modo como se explica a continuación.

Para pasar el valor hexadecimal 5AC directamente a binario, sólo tenemos que aislar cada una de las cifras y pasarla a binario, de forma que su valor en binario conste de 4 bits. Si no llega a tener 4 bits, tendremos que añadir bits a cero por la izquierda. Así obtenemos que 5 en binario es Ø1Ø1, A en decimal es 1Ø y en binario 1Ø1Ø; por último, C en decimal es 12 y en binario 11ØØ. Por lo tanto, podemos decir que el valor hexadecimal 5AC es Ø1Ø11Ø1Ø11ØØ.

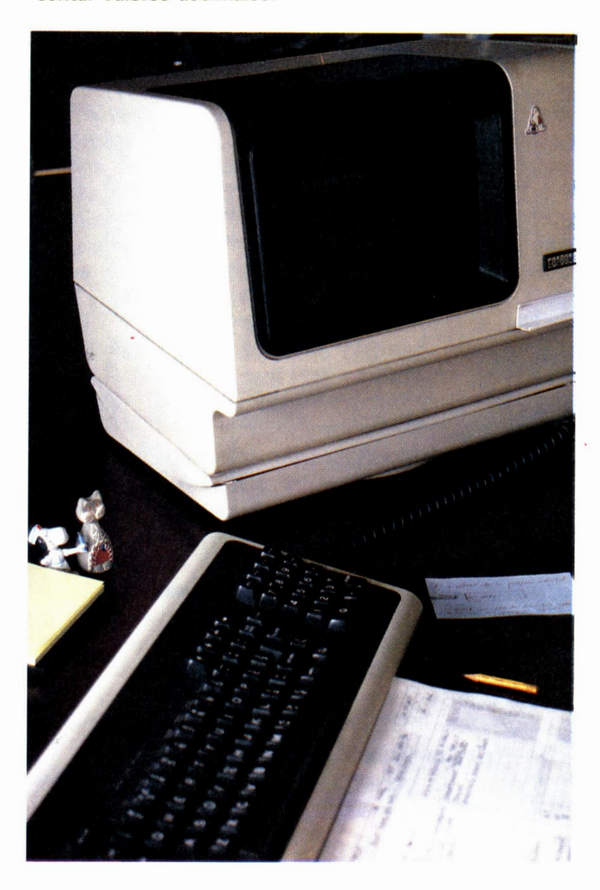
$$\underbrace{\text{Ø1Ø1}}_{5} \qquad \underbrace{\text{1Ø1Ø}}_{A} \qquad \underbrace{\text{11ØØ}}_{C}$$

Para realizar la comprobación, podemos pasar el valor binario Ø1Ø11Ø1Ø11ØØ a decimal, tal y como se ha descrito en los apartados anteriores. Así, $\emptyset \times 2^0 + \emptyset \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + \emptyset \times 2^4 + 1 \times 2^5 + \emptyset \times 2^6 + 1 \times 2^7 + 1 \times 2^8 + \emptyset \times 2^9 + a \times 2^{10} = 4 + 8 + 32 + 128 + 256 + 1.024 = 1.452$. Obtenemos el valor decimal 1.452.

Igualmente pasamos el valor 5AC hexadecimal a decimal, para ver si el resultado es el mismo. Así:

$C \times 16^0 + A \times 16^1 + 5 \times 16^2 = 12 \times 16^0 + 1\emptyset \times 16^1 + 5 \times 16^2 = 12 + 16\emptyset + 128\emptyset = 1452$

La computadora utiliza configuraciones de bits para representar valores decimales.



Obtenemos el valor decimal 1.452, que, como vemos, es el mismo que el obtenido al pasar de binario a decimal. Por lo tanto, el método que hemos utilizado para la conversión directa de hexadecimal a binario es correcto.

El segundo caso, es decir, la conversión de binario a hexadecimal, se efectúa realizando el proceso inverso al anterior, de tal manera que se divide el valor binario en grupos de 4 bits empezando por la derecha y después se convierte cada grupo a su valor hexadecimal.

Por ejemplo, supongamos que queremos pasar a hexadecimal el valor binario 11ØØ Ø1Ø1 1Ø1Ø. Tal como hemos dicho, reunimos los bits en grupos de 4 empezando desde la derecha. Así obtendremos 11ØØ, Ø1Ø1, 1Ø1Ø. A continuación, calcularemos el valor hexadecimal de cada uno de estos grupos de bits.

1100 = 12 = C, 0101 = 5, 1010 = 10 = A

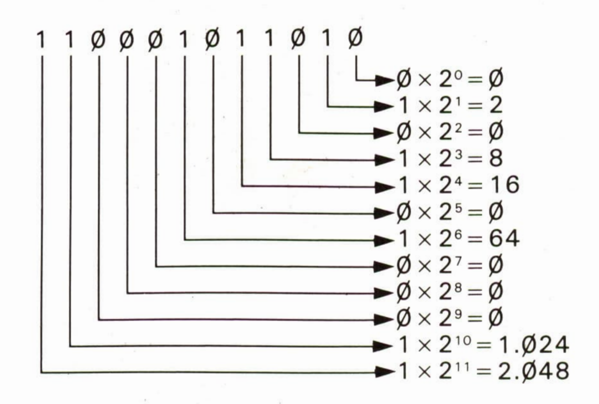
Obtenemos el valor hexadecimal C5A.

Si hacemos lo mismo que en el caso anterior, para comprobar si la conversión está bien hecha, pasando tanto el valor binario como el hexadecimal a decimal, obtenemos el mismo valor. Así:

C 5 A

A × 16⁰ = 1Ø × 1 = 1Ø
5 × 16¹ = 5 × 16 = 8Ø
C × 16² = 12 × 265 = 3.Ø72

1Ø + 8Ø + 3.Ø72 = 3.162



2 + 8 + 16 + 64 + 1.Ø24 + 2.Ø48 = 3.162

Efectivamente, 3.162 es decimal en los dos casos. Por lo tanto, la conversión se ha realizado bien.

Veamos un par de ejemplos más de estas conversiones para comprenderlas mejor.

Conversión del valor 2AB1 hexadecimal a binario:

2	A	B	1
ØØ1Ø	1Ø1Ø	1Ø11	ØØØ1

El valor binario será 10, 1010, 1011, 0001

Conversión del valor 10 1010 1011 0001 binario a hexadecimal:

ØØ1Ø	1Ø1Ø	1Ø11	ØØØ1
2	A	B	1

El valor hexadecimal será 2AB1.

Para entender también la conversión de los 4 bits a valor hexadecimal y del valor de una cifra hexadecimal a 4 bits, veamos todos los valores hexadecimales que pueden expresarse con 4 bits:

0000 = 0	1000 = 8
0001 = 1	1001 = 9
0010 = 2	1010 = A
0011 = 3	1011 = B
0100 = 4	1100 = C
0101 = 5	1101 = D
0110 = 6	1110 = E
0111 = 7	1111 = F

Siempre sabiendo que A = 10 decimal
B = 11
C = 12
D = 13
E = 14
F = 15

ORGANIZACIÓN DE LOS DATOS

Una computadora, por pequeña que sea, puede manejar gran cantidad de datos, pero a condición de que éstos estén organizados de una manera determinada para poder acceder a ellos de forma más fácil y rápida.

La información se encuentra almacenada en los dispositivos o unidades de almacenamiento externo de memoria, siendo en los soportes de estas unidades donde los datos están organizados de una manera especial.

Esta información se puede organizar de dos maneras: física y lógicamente. La organización física depende de los soportes sobre los que estará grabada la información y, por lo tanto, puede variar según las características de cada uno de estos soportes. En cambio, la organización lógica varía a voluntad del usuario, aunque las herramientas que éste tiene a su disposición para definir esta organización son fijas.

La diferencia entre estas dos organizaciones estriba en el hecho de que los datos no están grabados tal como un usuario los ve. Físicamente, los datos están grabados de una manera, pero el usuario se los imagina de otra. A pesar de esta apariencia, todo es correcto, ya que es el propio sistema operativo el que se encarga de realizar la transformación; es decir, el usuario trabaja con los datos (graba, lee, actualiza, etc.) mediante las herramientas que posee, pe-

ro siempre viéndolos organizados lógicamente, siendo el sistema operativo el que transforma estas órdenes para que dicha información quede alterada físicamente.

El usuario no tiene por qué preocuparse de la organización física, ya que, como hemos dicho antes, es el sistema operativo el que realiza la transformación de la organización lógica a la organización física. Por lo tanto, en este apartado solamente nos vamos a ocupar de la organización lógica de los datos.

Dentro de la organización lógica de los datos, las dos agrupaciones de información más importantes son los registros y los ficheros.

Ficheros y registros

Un fichero es un conjunto de elementos que contienen información relativa a un mismo tema y en el que cada uno de estos elementos es un registro.

Un registro forma parte de un fichero y a su vez está dividido en campos, de tal manera que cada campo contiene un dato referente a tal registro.

Por ejemplo, podemos encontrarnos ficheros que contienen información relativa a los artículos contenidos en un almacén, en los que cada registro contiene informaciones tales como el número de artículo, su nombre, la cantidad que hay de él en el almacén, la fecha de la última entrada de este artículo en el almacén, etcétera.

Registro físico y registro lógico

El registro lógico es el conjunto de información relativa a cada elemento de un fichero. El registro físico es la cantidad de información que puede leerse o grabarse de una sola vez y que depende de las características técnicas de la unidad de almacenamiento externo de memoria que contenga el soporte del que forma parte el fichero.

De esto se deduce que un registro físico puede contener varios registros lógicos (es lo más normal) o, al revés, un registro físico puede ser una parte de un registro lógico (no es normal); ello significa que, en el primer caso, en una operación de lectura física se leerán varios registros lógicos y, en el segundo caso, para leer un registro lógico habrá que realizar varias lecturas físicas.

Pongamos por caso el ejemplo anterior; sea la longitud del registro de cada artículo de 128 caracteres o bytes. Supongamos también que las características técnicas de la unidad de lectura hacen que cada vez que se haga una lectura física se lean 512 bytes, es decir, que la longitud del registro físico es de 512. De estas dos suposiciones se deduce que un registro físico constará de 4 registros lógicos (512/128 = 4) o, lo que es lo mismo, al efectuar una operación de lectura física sobre el fichero de los artículos contenidos en el almacén, se leerá de golpe la información correspondiente a 4 artículos (4 registros lógicos).

De la misma manera, si suponemos que la longitud del registro de cada empleado del almacén es de 1.024 caracteres o bytes, entonces, para leer la información correspondiente a un solo artículo, habrá que efectuar dos lecturas físicas.

Nótese que estos dos términos están relacionados con el de la lectura física y la lectura lógica. Si nos fijamos en los dos párrafos anteriores, veremos que se está utilizando mucho el término «lectura física». Este término significa lo que realmente quiere decir, es decir, un desplazamiento de los cabezales de lectura/grabación a una zona determinada del disco para obtener de golpe 512 bytes; se identifica con registro físico porque, como vemos, cada vez que se hace una lectura física se obtiene un récord físico.

Sin embargo, el término «lectura lógica» significa cualquier orden de lectura que el programador o usuario especifique en forma de instrucción dentro de un programa y que no coincida con una lectura física, sino que sea la lectura de un registro lógico.

Pensemos entonces que, por cada 4 lecturas lógicas en forma de instrucción que efectuará un programa, solamente se efectuará una de física. Para comprenderlo mejor véase el cuadro **lectura física y lectura lógica** *(página 214)*.

El usuario es completamente ajeno a este proceso y sólo ve que, cuando su programa efectúa una lectura, obtiene el registro de un artículo.

Organización física y organización lógica

Por ejemplo, cuando en un disco tenemos información grabada referida a una serie de artículos, nos imaginamos una información referida a un artículo, grabada inmediatamente después de la del artículo anterior y así sucesivamente. Sin embargo, físicamente no es así; puede ser que la información de un artículo esté grabada físicamente al final del disco y la del siguiente artículo esté grabada al principio del disco, aunque, al obtener un listado de tales datos, nos salgan los artículos correctamente uno a continuación del otro.

Lectura física y lectura lógica

10 READ LECTURA LÓGICA DEL REG. 1 20 READ LECTURA LÓGICA DEL REG. 2 30 READ LECTURA LÓGICA DEL REG. 3 40 READ LECTURA LÓGICA DEL REG. 4	→ LECTURA FÍSICA	Reg. 1 Reg. 2 Reg. 3 Reg. 4
50 READ LECTURA LÓGICA DEL REG. 5 60 READ LECTURA LÓGICA DEL REG. 6 70 READ LECTURA LÓGICA DEL REG. 7 80 READ LECTURA LÓGICA DEL REG. 8	→ LECTURA FÍSICA	Reg. 5 Reg. 6 Reg. 7 Reg. 8
90 READ LECTURA LÓGICA DEL REG. 9	→ LECTURA FÍSICA	Reg. 9 Reg. 10 Reg. 11 Reg. 12
		⋮

Campos de un registro

Los registros de un fichero, como ya hemos dicho antes, se dividen en campos, cuyo contenido se corresponde con cada dato específico de la información a la que representa el registro.

Existen varias clases de campos atendiendo a su contenido o a su función dentro del registro.

Si tomamos en cuenta su contenido, los campos pueden ser:

- *Numéricos*, si sólo contienen números.
- *Alfanuméricos*, si contienen números, letras y otros signos de puntuación y especiales.

Atendiendo a la función del campo dentro del registro, podemos tener:

- *Campos claves.*
- *Campos normales.*

Los campos claves son campos numéricos o alfanuméricos que sirven para identificar al registro. En el ejemplo anterior, en el fichero cuyos registros son la información de cada artículo podemos tener un campo clave, que es el número de artículo, que sirve para identificar a cada uno de ellos. Los normales son todos los demás campos que no sean claves.

Existen también campos que se subdividen en otros que reciben el nombre de zonas. Pongamos por ejemplo el campo «fecha última entrada». Este campo se subdivide en tres zonas, que serán el día, el mes y el año.

Clases de ficheros

Podemos establecer dos grandes grupos de ficheros, según su función y según la organización de sus registros.

Los ficheros se dividen en tres tipos según la función que desempeñan:

— ficheros maestros;
— ficheros de trabajo;
— ficheros de movimientos.

FICHEROS MAESTROS

Son ficheros permanentes, es decir, que, una vez creados, sufren la introducción de nuevos registros, la modificación de los ya existentes o la anulación de algunos de ellos y en ningún caso se destruyen para volverlos a crear posteriormente.

La información que almacenan es muy importante y normalmente se necesita consultar muy a menudo.

Ejemplos de ficheros maestros son: un fichero de cuentas en una aplicación de contabili-

dad, un fichero de artículos en una aplicación de facturación, un fichero de títulos de películas en una aplicación de vídeo clubs, un fichero de libros en una aplicación bibliotecaria.

FICHEROS DE TRABAJO

Son ficheros cuya vida es muy corta. Normalmente, como su nombre indica, sirven de trabajo, es decir, de medio para obtener otros ficheros mucho más importantes, como pueden ser los maestros. La información que almacenan es importante, pero solamente durante un corto espacio de tiempo.

Un ejemplo de este tipo de ficheros es el fichero de trabajo que se crea y utiliza para obtener otro fichero maestro clasificado por una clave concreta. En general, cualquier fichero que se crea en un programa y que al final de ese programa puede ser anulado, pertenece a este tipo.

FICHEROS DE MOVIMIENTOS

Son ficheros que pueden ser permanentes, pero que también pueden no serlo. Se trata de ficheros que sufren muchas altas y muy pocas bajas. Normalmente, la información que contienen es información de consulta o información que más tarde será utilizada por otro programa para ayudar en la construcción de otro fichero de movimientos o para actualizar los registros de un fichero maestro. Es decir, que pueden contener informaciones de dos tipos: de consulta y de trabajo. Los que contienen información de consulta, si no sobrepasan la capacidad que tienen asignada, no serán destruidos; pero, si la sobrepasan, serán destruidos para volver a generarlos con nueva información. La vieja información que contenían normalmente se imprime en papel y se guarda archivada en tal papel. Los que contienen información para actualizar otro fichero siempre son destruidos una vez producida tal actualización.

Ejemplo de ficheros de movimiento son: un fichero de apuntes contables en una aplicación de contabilidad, un fichero de extractos en la misma aplicación, un fichero de apuntes de facturación para realizar el diario de facturación en una aplicación de facturación.

Los ficheros también se dividen en tres tipos, según la manera en la que están organizados sus registros:

— ficheros de organización directa o al azar;
— ficheros de organización secuencial;
— ficheros de organización secuencial indexada.

FICHEROS DE ORGANIZACIÓN DIRECTA O AL AZAR

Un fichero puede estar ordenado mediante un campo clave presente en cada uno de sus registros para facilitar el acceso a ellos; sin embargo, es importante tener en cuenta que en este tipo de organización no es necesario que el fichero esté clasificado.

Nos encontramos con un fichero de organización directa cuando existe una relación entre las claves de los registros y las posiciones que ocupan éstos dentro del fichero.

Estos ficheros, según la forma de acceso a sus registros, se dividen en muchas clases; sin embargo, únicamente describiremos las dos más sencillas y más usadas:

— ficheros de direccionamiento directo;
— ficheros de direccionamiento por tabla.



La imagen muestra una aplicación computacional que empieza a ser bastante corriente en las consultas médicas, en este caso el consultorio de un pediatra. Mediante un fichero maestro se van almacenando los nuevos datos de cada consulta o nueva exploración del niño o bebé, modificando los ya existentes o anulando algunos de ellos, pero sin destruirlos en ningún caso, por la importancia que poseen para conocer en todo momento la evolución del historial clínico, así como el curso de las posibles enfermedades o trastornos diagnosticados.

Ficheros de direccionamiento directo

En este tipo de organización, la clave coincide con la posición que ocupa el registro que contiene esta clave dentro del fichero. Es decir, si suponemos como de acceso directo el fichero de artículos descrito anteriormente y utilizamos el campo «n.º de artículo» como campo clave, resultará que el registro 1 se corresponderá con el del n.º del artículo 1, el registro 2, con el del n.º del artículo 2 y así sucesivamente.

	N.º ARTÍCULO	NOMBRE	
REGISTRO 1	00001	Tijeras	
REGISTRO 2	00002	Agujas	
REGISTRO 3	00003	Dedales	
REGISTRO 4	00004	Cinta métrica	
⋮	⋮		

Para encontrar un registro que contenga un número de artículo determinado, por ejemplo el n.º 00003, se leerá el registro 3 y se encontrará tal registro. Uno de los inconvenientes de esta organización radica en que la clave tiene que ser forzosamente numérica.

Ficheros de direccionamiento por tabla

En este tipo de organización se construye una tabla a través de la cual se puede acceder directamente al registro del fichero que se desee. El método consiste en asociar cada clave con la posición del registro, que contiene esta clave dentro del fichero, y agrupar todas estas asociaciones en una tabla.

Supongamos el ejemplo de la columna siguiente .

	N.º ARTÍCULO	
REGISTRO 1	00001	
REGISTRO 2	01001	
REGISTRO 3	01510	
REGISTRO 4	00200	
REGISTRO 5	10250	

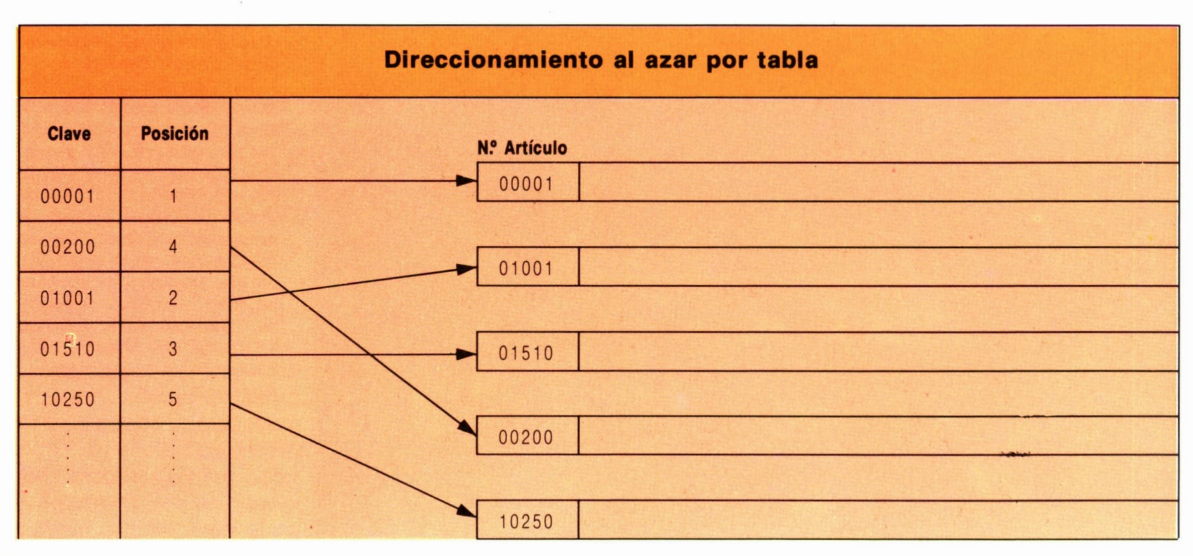
Dado este fichero de artículos, para que su organización sea de **direccionamiento al azar por tabla** habrá que construir una tabla como la especificada en el cuadro homónimo.

Como vemos en esta tabla, las claves están clasificadas de mayor a menor. Para leer un determinado registro, por ejemplo, el de la clave o n.º del artículo 00200, primero se accederá a la primera parte de la tabla para encontrar dicho n.º de artículo (si están clasificadas, existen métodos para encontrarlo rápidamente). Una vez encontrado, se leerá la posición asociada a tal clave o número, que en este caso será 4. Con esto ya sabremos en qué registro se encuentra el n.º del artículo 00200, con lo que bastará leer tal registro, el número 4.

Ficheros de organización secuencial

Los ficheros de organización secuencial son aquellos en los que sus registros están almacenados consecutivamente y en el mismo orden en el que se grabaron.

Son ficheros en los que la única forma de acceder a ellos es secuencialmente, es decir, accediendo al primer registro y después sucesi-



vamente al 2.º, 3.º, 4.º, etc. De esta manera, si se quiere acceder a un determinado registro de uno de estos ficheros, la única manera de llegar a él consiste en leer todos los registros, empezando por el 1.º, hasta dar con el deseado.

Como vemos, estos ficheros no sirven como ficheros maestros, sino más bien como ficheros de trabajo, ya que su organización hace que su acceso y actualización sean, en la mayoría de los casos, muy lentas.

Normalmente, siempre se suelen tener clasificados, ya que, de este modo, al acceder secuencialmente a ellos, sabemos que el primer registro contendrá el campo más pequeño o más grande (depende de la clasificación ascendente o descendente) por el que se ha realizado la clasificación, el 2.º registro contendrá el segundo campo más grande o más pequeño, etcétera.

Un ejemplo de fichero de organización secuencial lo podemos ver en una aplicación de facturación. Se trata de realizar un impreso de manera que en cada línea figuren los datos de cada factura que se ha elaborado en el propio proceso de facturación.

En el momento de elaborar las facturas, como la impresora está ocupada con esta elaboración, no puede al mismo tiempo realizar el impreso antes descrito. Entonces, lo que se puede hacer es almacenar las informaciones correspondientes a cada factura en cada registro de un fichero secuencial. De este modo, una vez acabada la facturación, leyendo este fichero de manera secuencial (empezando desde el primer registro al último) e imprimiendo por cada lectura la información del registro, obtendremos el impreso deseado.

Tal fichero puede clasificarse por el número de factura o por el campo que se quiera; entonces el listado se obtendrá con las líneas clasificadas de menor a mayor o de mayor a menor con respecto al campo por el que se ha realizado la clasificación.

Este tipo de ficheros no se utilizan muy a menudo.

Ficheros de organización secuencial indexada

Un fichero de organización secuencial indexada consta de tres áreas:

— un área de índices o claves;
— un área de datos;
— un área de overflow.

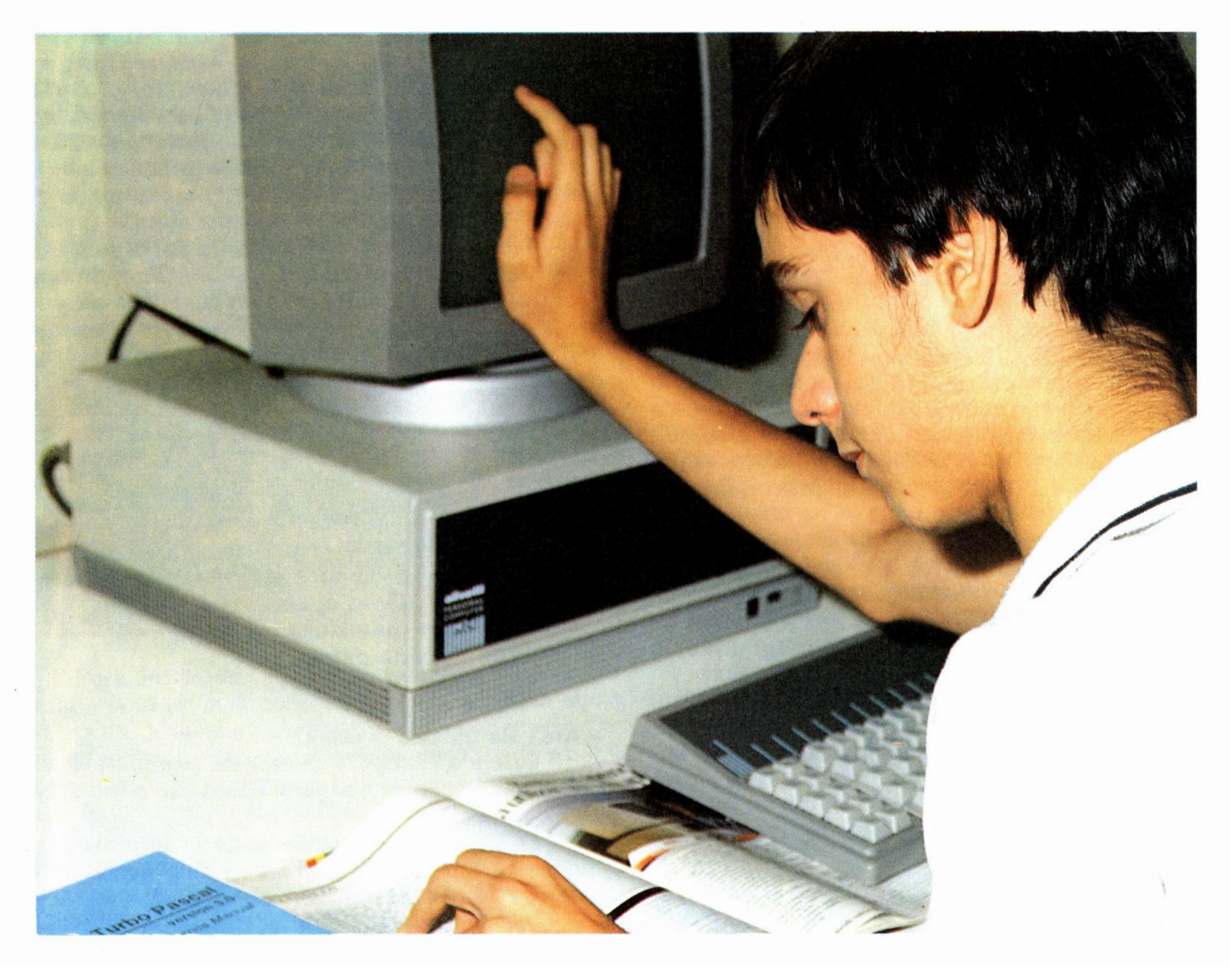
Área de índices o claves

Esta área está compuesta de entradas, cada una de las cuales contiene una clave o índice y una dirección.

La descripción detallada de la información que se encuentra en esta área y su organización es muy difícil de explicar; para hacer más fácil la explicación, solamente diremos que contiene



Sala de archivos de una microteca, donde se albergan las máquinas para la impresión de las microfichas, imprescindibles para la mecanización de una biblioteca computerizada.



El Pascal es un lenguaje relativamente sencillo cuyo aprendizaje puede iniciarse a edad temprana. En la fotografía, un estudiante verifica un programa, correspondiente a un ejercicio de biología, escrito en este lenguaje.

información de la clave más alta que hay en cada cilindro y de la clave más alta que hay en cada pista, de tal manera que siempre se sabe, al entrar un nuevo registro o clave, dónde le corresponderá entrar, es decir, en qué cilindro o pista.

El manejo de esta área viene soportado por el sistema operativo.

Área de datos

En esta área se almacenan los registros, es decir, la información más importante. Estos registros se grabarán automáticamente clasificados en orden ascendente de la clave. Esta grabación automática la realiza el propio sistema, porque antes ha pasado por el área de índices y de ahí ha tomado la información que le dice dónde debe grabar el registro para que quede bien clasificado.

Lógicamente tanto las operaciones de anulación de un registro como las de lectura y actualización siguen el mismo camino; una vez especificada la clave que quiere leerse, anularse o actualizarse, el sistema consulta en el área de índices, donde obtiene información física sobre la pista/cilindro en que se encuentra el registro perteneciente a esa clave; a partir de ahí, lo puede anular, leer o modificar.

Área de overflow

Es posible que en un momento determinado el espacio físico destinado al área de datos se agote; entonces, para que un usuario no se quede a mitad de trabajo, los registros que deberían grabarse en el área de datos se grabarán en un área del mismo fichero llamada overflow. Esta situación quedará reflejada en el área de índices.

Más tarde puede realizarse una reorganización del fichero, con lo cual se consigue dar más espacio al área de datos y trasladar los registros que se encuentran en el área de overflow al lugar que les corresponde en el área de datos. Por lo tanto, vemos que esta área es sólo para uso transitorio, para que el fichero, en un momento determinado, no haga que el programa que lo está utilizando se interrumpa a causa de un error al no haber más espacio en el área de datos.

Lenguajes de programación

Los lenguajes de programación son los medios de comunicación entre los programadores o usuarios y la computadora. Con ellos se construyen los programas que luego serán ejecutados por la computadora.

Existen dos categorías diferentes de lenguajes:

— los lenguajes de bajo nivel;

— y los lenguajes de alto nivel.

Los lenguajes de bajo nivel son los *lenguajes assembler* y los *lenguajes máquina*. En estos lenguajes, cada línea de programa que se escribe con ellos es una orden para la computadora; en el caso de los lenguajes assembler, cada una de estas líneas se construye con unas instrucciones específicas, cada una de las cuales tiene un nombre. Sin embargo, con el lenguaje máquina, las instrucciones o líneas de programas son cadenas de 0 y 1 o de caracteres hexadecimales, con lo que resulta muy difícil y complicado programar directamente con él. Hoy en día, la programación en lenguaje máquina ya no se realiza, dada la existencia de los lenguajes assembler (aunque éstos sólo se utilizan en determinadas ocasiones) y los lenguajes de alto nivel.

Tanto los lenguajes assembler como los de alto nivel, antes de ser ejecutados, se traducen a lenguaje máquina, único lenguaje interpretable por la computadora. En el caso de los assembler, su traducción a lenguaje máquina se efectúa mediante un programa llamado ensamblador, y los de alto nivel se traducen mediante compiladores e intérpretes.

Cada instrucción creada con un lenguaje de alto nivel consta de varias órdenes para la computadora. Es decir, así como en el lenguaje assembler o máquina, para realizar la suma de dos variables, es posible que necesitemos varias instrucciones, en un lenguaje de alto nivel con una sola instrucción basta, ya que luego, al ser compilada, esta instrucción se transformará automáticamente en varias instrucciones en lenguaje máquina, que serían las mismas que si estuviéramos programando en assembler o en lenguaje máquina.

Existe un gran número de lenguajes de alto nivel y cada vez aparecen lenguajes de este tipo más evolucionados, que hacen que un usuario con poca experiencia pueda empezar a programar una computadora con cierta facilidad.

En estos lenguajes, el programador no debe preocuparse del funcionamiento interno de la computadora (unidad central de proceso, registros de la CPU, posiciones físicas de memoria, etc.), cosa que no ocurre en los lenguajes assembler y máquina, que constantemente exigen trabajar con estos elementos.

LOS LENGUAJES ASSEMBLER

Como ya hemos dicho en la introducción, en los programas escritos con lenguaje assembler cada instrucción o línea de programa se corresponde con una acción que debe ejecutar la computadora, al igual que en los programas escritos en lenguaje máquina. Pero los lenguajes assembler, en lugar de dar un código numérico a una instrucción, representan la instrucción mediante símbolos (letras). Por ejemplo, para indicar la instrucción «suma» se hace, en algunos de ellos, con el carácter *A,* que es el primer carácter de la palabra inglesa *add,* que significa suma o sumar.

Los programas escritos en lenguaje assembler ocuparán menos espacio, una vez ensamblados y traducidos a lenguaje máquina, que los de alto nivel una vez compilados. Incluso en algunos casos se ejecutarán más rápidamente, siempre que estén bien construidos y optimizados.

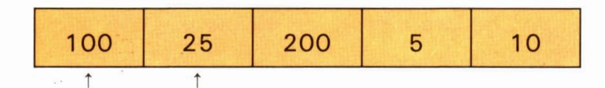
Para comprender mejor la relación entre los lenguajes assembler y los lenguajes máquina, en la página 226 se muestra un programa escrito en lenguaje assembler y en la página 227 aparece el correspondiente en lenguaje máquina.

Este programa realiza la clasificación de una cierta cantidad de números, especificada por el usuario fuera del programa, mediante el método de la burbuja, el cual se explica a continuación.

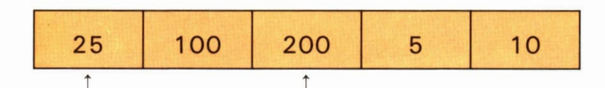
Dado un grupo de cantidades numéricas que se encuentran en una tabla totalmente desclasificadas, se trata de obtener tal tabla de números clasificada de menor a mayor o mediante la comparación de todos los números con el primero. En cada comparación, si el número comparado es menor que el que ocupa la primera posición, entonces se intercambian la posición hasta llegar a comparar el primero con el últi-

mo, después de lo cual en la primera posición tendremos el número menor de la tabla. Seguiremos después comparando el de la segunda posición con todo el resto y, al llegar al final, en el segundo lugar tendremos el más pequeño después del primero. Haciendo este mismo proceso tantas veces como posiciones haya, al final obtendremos la tabla clasificada.

Ejemplo. Tenemos la tabla de números siguiente y la queremos clasificar:



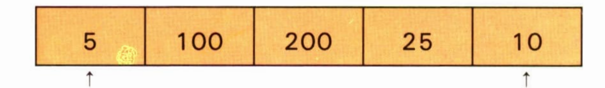
Comparamos 100 con 25 y, como 100 es mayor que 25, los cambiamos de lugar.



Comparamos 25 con 200 y, como 25 es menor que 200, no cambiamos.



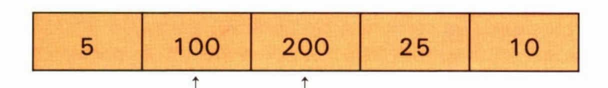
Comparamos 25 con 5 y, como 25 es mayor que 5, los cambiamos de lugar.



Comparamos 5 con 10 y, como 5 es menor que 10, no cambiamos.



Efectuadas ya todas las comparaciones con la primera posición de la tabla, vemos que, realmente, en la primera posición está el número más pequeño.



Empezaremos ahora a comparar el segundo con el tercero, es decir, 100 con 200; como 100 es menor que 200, no cambiamos.



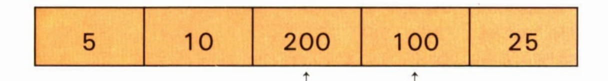
Comparamos 100 con 25 y, como 100 es mayor que 25, los cambiamos de lugar.



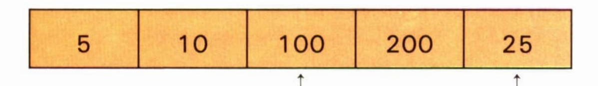
Comparamos 25 con 10 y, como 25 es mayor, lo cambiamos de lugar.



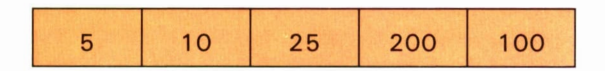
Efectuadas ya todas las comparaciones con el segundo elemento, en este lugar se encuentra el segundo número más bajo de la tabla.



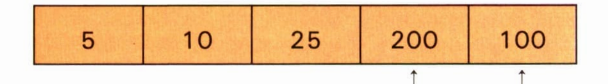
Empezaremos ahora a comparar el tercero de los elementos con el cuarto, es decir, 200 con 100; como 200 es mayor, los cambiaremos de lugar.



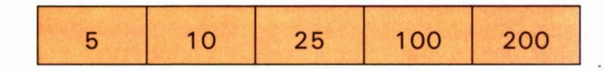
Comparamos 100 con 25 y, como 100 es mayor, los cambiamos de lugar.



Una vez efectuadas todas las comparaciones con el tercer elemento, en este lugar nos encontramos con el tercer número más bajo.



Comparamos a continuación el cuarto y el quinto, es decir, 200 con 100; como 200 es mayor, los cambiaremos de lugar.



Al no haber ya más posiciones por comparar, hemos llegado al final y, como podemos ver, la tabla ha quedado clasificada. Estos mismos pasos que hemos efectuado con esta tabla se efectúan en el programa assembler mostrado anteriormente.

En la página 227 podemos ver el programa de assembler escrito en lenguaje máquina. Cada par de números representa un byte y están codificados en hexadecimal.

LOS LENGUAJES DE ALTO NIVEL

Aunque existen más de cien lenguajes de alto nivel, solamente vamos a ocuparnos de los más importantes o los que son usados por un número mayor de usuarios o programadores. Éstos son el lenguaje COBOL, el FORTRAN, el PASCAL, el PL/1, el ADA, el LOGO y, por último, el lenguaje más conocido y el más utilizado, el BASIC. Este último lenguaje merece un capítulo aparte debido a su importancia; en dicho capítulo se intenta realizar una descripción lo más completa y comprensible que se pueda.

El lenguaje COBOL

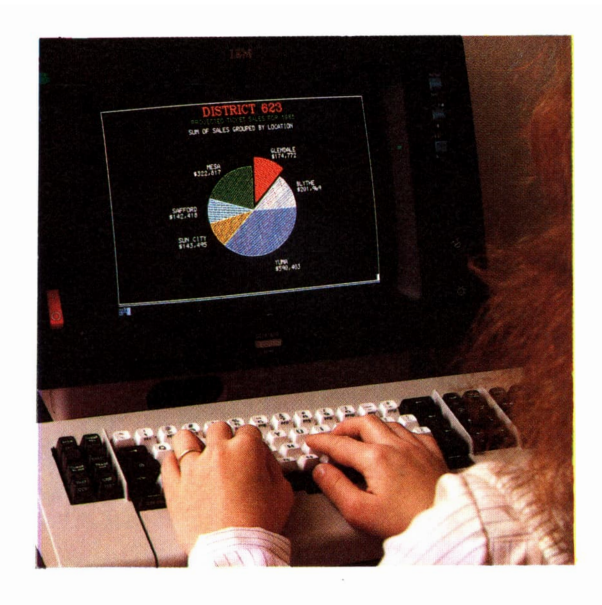
La palabra COBOL la conforman las siglas de *Common Business-Oriented Language,* que significa «lenguaje de programación orientado a los problemas de gestión». Fue concebido en el año 1959 y empezó a difundirse en los primeros años de la década de los sesenta. Surgió ante la necesidad de un lenguaje puramente dedicado a resolver problemas de gestión, ya que los que había hasta entonces estaban más orientados a resolver problemas de carácter científico y numérico.

Este lenguaje se concibió para ser independiente de la computadora sobre la que se haya implementado; es decir, si tenemos una aplicación de gestión escrita en COBOL, que funciona correctamente en una determinada computadora con muy pocos cambios en sus programas, deberá funcionar en otra computadora diferente de la primera y que también pueda trabajar en COBOL.

Un programa escrito en COBOL normalmente es muy largo, aunque el trabajo que tenga que hacer sea muy corto. Éste es uno de los inconvenientes que tiene este lenguaje.

Permite nombres de variables de hasta 30 caracteres de longitud y sus sentencias o instrucciones normalmente están formadas por palabras inglesas, por ejemplo, ADD (suma), SUBTRACT (resta), etcétera.

Cuando se construye un programa en lenguaje COBOL, tal programa obligatoriamente estará dividido en 4 secciones, que serán las siguientes.



Arriba, operadora tecleando un programa de gestión en lenguaje COBOL, que ofrece la ventaja de poder homologarse en cualquier computadora.

- *Identification Division.* En ella figuran sentencias en las que se identifican el nombre del programa, el del autor del programa y otros datos de identificación del programa. Como se comprenderá, esta sección solamente se crea a efectos de documentación.
- *Environment Division.* En ella, a los nombres de los ficheros con los que se va a trabajar se les define su ubicación física, es decir, en qué disco, cinta, etc., se encuentran; incluso se indica que los ficheros de salida se imprimirán en impresora.
- *Data Division.* En ella se definen todas las variables y todas las áreas de trabajo que se van a utilizar en el programa, así como la relación entre ellas.
- *Procedure Division.* En ella se crea, en realidad, el programa con sentencias de acción; no de definición, como en las secciones anteriores.

En la sección *environment division* será donde se producirá algún cambio al trasladar cualquier programa de una computadora a otra, ya que será en las unidades de entrada/salida donde puede estar la incompatibilidad.

Al estar este lenguaje orientado a resolver problemas de gestión, los programas construidos con él manejarán ficheros con mucha información, es decir, ficheros muy grandes, y, por lo tanto, debe tener la posibilidad de definir ficheros con una organización potente; efectivamente, así es. Normalmente, el lenguaje COBOL permite definir y tratar ficheros con cualquier organización, incluso con organización secuencial indexada, que es la que se utiliza para los ficheros maestros.

Como es lógico, este lenguaje no es óptimo para crear programas que tengan que resolver problemas de tipo científico o técnico.

En las páginas 227 y 228 se muestra un programa escrito en COBOL.

El programa escrito en COBOL realiza una lectura registro a registro de un fichero, visualizando en la pantalla de la computadora la información de tal registro y preguntando si es correcto o no. Si es correcto, en uno de sus campos se introduce la palabra «correcto» y se vuelve a grabar en el fichero. Si no es correcto, en el mismo campo se introduce la palabra «no correcto» y se vuelve a grabar.

Como podemos ver, en la 1.ª sección (la *identification division*) definimos el nombre del programa, «EJEMPLO», y a continuación la fecha de escritura de tal programa, «OCT-85».

En la siguiente sección, la *data division*, definimos todas las variables con las que vamos a trabajar y los campos del récord del fichero FICH-MOV. La descripción de los campos del récord se hace desde la línea 23 a la 31.

De las líneas 37 a la 45 y de la 51 a la 58 se definen los campos de la pantalla, títulos y valores que estarán en relación con los campos del récord leído.

En la siguiente sección es donde realmente se ejecuta el programa, ya que en ella se encuentran las órdenes ejecutivas. En dicha sección se lee un registro, se visualizan sus campos en la pantalla, se pide al usuario que conteste desde teclado si la información es correcta; si es así, se introduce en un campo de tal registro la palabra «correcto» y se graba el registro; si no es correcto, se introduce la palabra «no correcto» y se graba igualmente. Una vez aceptada o no tal información, se vuelve a leer otro registro y así sucesivamente hasta que se llegue al final del fichero, con lo que terminará la ejecución del programa.

INFORMACIÓN DE UN REGISTRO EN PANTALLA

La visualización de la información de un registro en la pantalla es de esta forma.

Supongamos el registro anterior:

El lenguaje FORTRAN

El nombre de este programa viene de las palabras *FORmula TRANslator,* que significa traducción de fórmulas. Surgió en el año 1954 y es el lenguaje de alto nivel más antiguo que se utiliza aún hoy día. Fue diseñado por científicos, precisamente para resolver sus problemas; por lo tanto, está orientado a la resolución de problemas científicos y técnicos.

Este lenguaje no tiene instrucciones de acceso a ficheros o de entrada/salida muy potentes, ya que no está hecho, como el COBOL, para manejar mucha información almacenada en ficheros; más bien está concebido para poder realizar muchos cálculos de una manera muy rápida y, por lo tanto, las instrucciones más potentes que tiene son las que realizan estos cálculos.

Se utiliza mucho en universidades para realizar trabajos que requieran muchos cálculos e incluso tiene la posibilidad de trabajar en doble precisión, lo que le permite dar resultados con muchos más dígitos significativos que trabajando normalmente.

Un programa escrito en FORTRAN es mucho más difícil de seguir que otros programas escritos en otros lenguajes, como por ejemplo el COBOL.

En la página 229 puede verse un ejemplo de un programa escrito en FORTRAN.

En dicho programa se realiza el cálculo del área de un trapecio. Entrando como datos el número del trapecio, la longitud de sus dos bases y de la altura, la computadora calcula el área. En este caso le hemos entrado datos referentes a tres trapecios, que son los siguientes:

Trapecio n.º	Base 1	Base 2	Altura
1	4	4	2
2	6	2	3
3	10	4	4

y la computadora nos ha devuelto los resultados que se muestran en el programa FORTRAN de la página 229.

Este programa es muy sencillo, ya que, en realidad, la única sentencia de cálculo es la que sigue a la 10. Las demás instrucciones son de lectura y escritura.

PEPITA JIMÉNEZ .... 54321 CAMELIAS .... 55 .... REGALO ....
FLORERO CHINO .... 23451 .... 90.000 ....

CLIENTE	COD. CLIENTE	DIRECCIÓN	
PEPITA JIMÉNEZ	54321	CAMELIAS 55	
ARTÍCULO	**COD. ARTÍCULO**	**DESCRIPCIÓN**	**COSTE**
REGALO	23451	FLORERO CHINO	90.000



Arriba, vídeo juegos en pantalla, realizados mediante un lenguaje de alto nivel.

El lenguaje PASCAL

Este lenguaje recibe su nombre del matemático y filósofo francés Blaise Pascal y fue desarrollado por Niklaus With en el año 1970.

Puede decirse que este lenguaje es un intermedio entre el FORTRAN y el BASIC, ya que posee la potencia de cálculo y la rapidez del FORTRAN y la capacidad de sencillez del BASIC, aunque no tiene la potencia del manejo de grandes masas de información, como el COBOL. Además, es un lenguaje estructurado; es decir, los programas escritos con él son fáciles de seguir gracias a que su programación es secuencial y sin saltos de una línea a otra, los cuales siempre dificultan el seguimiento de un programa. El PASCAL es uno de los lenguajes con más poder de estructuración que existen.

Gracias a que posee lo mejor del BASIC y del FORTRAN, es un lenguaje que se ha adoptado para realizar programas en muchas computadoras que necesitan resolver tanto problemas científicos como problemas con poco cálculo, pero que precisan de una gran rapidez. Su único punto negro es la falta de potencia para manejar eficazmente grandes masas de información almacenadas en discos o cintas.

En la página 228 podemos ver un ejemplo de un programa escrito en PASCAL.

Los lenguajes PL/1 y ADA

Estos dos lenguajes no son tan importantes como los anteriores debido a algunas limitaciones en su uso; por lo tanto, nos limitaremos a citar algunas de sus particularidades.

El lenguaje PL/1 es un intermedio entre el FORTRAN y el COBOL; es decir, posee la potencia de cálculo del FORTRAN y la capacidad de manejo de grandes masas de información del COBOL. Como vemos, es un lenguaje muy potente, que fue introducido a mitad de la década de los sesenta por IBM; de ahí su limitación, ya que, en un principio, solamente se podía ejecutar en computadoras de esta marca, puesto que los otros fabricantes sólo fueron capaces de implantarlo en sus computadoras muy lentamente.

Por lo tanto, es un lenguaje muy potente que sirve tanto para resolver problemas científicos como problemas de gestión. Además, también es uno de los lenguajes con más poder de estructuración que existen hoy día.

El lenguaje ADA surgió en los años setenta ante la necesidad de los servicios de computación del departamento de defensa de Estados Unidos de programar en un lenguaje de alto nivel, ya que hasta entonces se programaba siempre en lenguajes de bajo nivel (assembler). Sin embargo, cuando se decidió, las diferentes ramas de estos servicios no se pusieron de acuerdo en el lenguaje de alto nivel que querían elegir y, finalmente, decidieron construir uno que colmara los requerimientos de todas las personas que intervenían en el desarrollo de programas.

Al fin, en el año 1980 se acabó de desarrollar un lenguaje de alto nivel, caracterizado fundamentalmente por su poder de estructuración, y al cual se le dio el nombre de ADA, en homenaje a Ada Augusta Lovelace Byron, compañera de Charles Babbage.

Este lenguaje debe su importancia a su parecido con el PASCAL y a su poder de estructuración.

El lenguaje LOGO

El LOGO es un lenguaje utilizado normalmente en las escuelas de enseñanza primaria, fundamentalmente para introducir a los escolares en el manejo y empleo de las computadoras.

Su función es muy diferente a la de los anteriores lenguajes que hemos visto, de ahí que su forma de trabajar sea completamente diferente a la de los demás lenguajes.

Fue desarrollado por el profesor Seymour Papert en el laboratorio de Inteligencia Artificial del Instituto Tecnológico de Massachussets, y su origen descansa en la búsqueda de métodos de aprendizaje en el manejo de las computadoras por parte de los niños.

Como es lógico, al ser un lenguaje fundamentalmente destinado a los niños, posee una sencillez y claridad asombrosas. Sin embargo, ello no quiere decir que no se puedan resolver problemas complicados con él, sino que más bien ahí radica su potencia; es decir que, siendo un lenguaje tan fácil de aprender y usar, pueden llegar a construirse programas muy complejos. Desde luego, no posee la potencia de un lenguaje COBOL, FORTRAN o PASCAL, pero no la necesita al no tener el mismo campo de actuación que ellos.

Tiene la ventaja de que no es preciso tener ningún conocimiento previo de computación ni de computadoras para aprender y construir programas con LOGO. Simplemente, basta con el aprendizaje de unos cuantos comandos para iniciar la construcción de algún programa que funcione.

Normalmente, los niños se introducen en este lenguaje a través de una de sus partes, tal vez la más didáctica para el aprendizaje: la construcción de gráficos mediante LA TORTUGA *(véase pág. 230)*.

LA TORTUGA es un pequeño triángulo que se visualiza en la pantalla y cuyo movimiento por ella se puede controlar mediante comandos y órdenes muy sencillas LOGO. Cuando un niño aprende a controlar el movimiento de LA TORTUGA, ya puede decir que está programando en LOGO.

Con los desplazamientos de LA TORTUGA se pueden dibujar complejas figuras geométricas, con lo que se introduce al niño en un mundo matemático o geométrico mediante el uso de la computadora. Sin embargo, el LOGO no es solamente LA TORTUGA, sino que también permite la resolución de problemas matemáticos, el manejo de listas, el uso de procedimientos, el uso de operaciones lógicas; además, posee comandos de entrada/salida y una característica muy importante, la recursividad.

Fotograma del film* Juegos de guerra*, película que plantea el gravísimo problema del acceso y su ulterior manipulación a los bancos de datos y a los programas secretos por personas extrañas y ajenas al servicio, en este caso el Departamento de Defensa de Estados Unidos.



PROGRAMA ESCRITO EN LENGUAJE ASSEMBLER

```
            Module  Clasificación, segmented
            Section Sect 1
            title   "Clasificación por la burbuja"
*
*
            dd      ØØ
            jr      Principio
*
msj         ddb     "ASM-CLAS ", %Øa, %Ød, ØØ
msj1        ddb     "Comienzo ", %Øa, %Ød, ØØ
msj2        ddb     "fin ", %Øa, %Ød, ØØ
msj3        ddb     "Número de parámetros no válido ", %Øa, %Ød, ØØ
msj4        ddb     "Número de elementos: ". ØØ
*
Principio   ldl     rr12,£ msj1
            sc      £89
            pop     r1,§rr14
            cp      r1,£2
            jr      NZ,error
            popl    rr6,§rr14
            ld      r3,§rr6
            ld      r2,£%Ø2ØØ
            popl    rr4,§rr14
            ld      r6,§rr4
            ldl     rr12,£msj4
            sc      £89
            ld      r12,r6
            ld      r13,£6
            sc      £94
            sc      £90
*
            add     r6,r6
*
lab2        ld      r1Ø,£Ø
            dec     r6,£2
            cp      r6,£2
            jr      LT,fin
            ld      r7,£Ø
            ld      r8,rr2(r7)
lab1        inc     r7,£2
            ld      r9,rr2(r7)
            cp      r8,r9
            jr      GT,intercambio
lab3        ld      r8,r9
            cp      r7,r6
            jr      ME,lab1
            cp      r1Ø,£Ø
            jr      Z,fin
            jr      lab2
intercambio ld      r1Ø,£1
            ex      r8,r9
            dec     r7,£2
            ld      rr2(r7),r8
            inc     r7,£2
            ld      rr2(r7),r9
            jr      lab3
*
*
*
error       ldl     rr12,£msj3
            sc      £89
```

```
               cp        r1,£Ø
               jr        EQ,fin

lab4           popl      rr1Ø,§rr14
               djnz      r1,lab4
               ret
*
*
*
fin            ldl       rr12,£msj2
               sc        £89
               sc        £90
               ret
```

PROGRAMA ESCRITO EN LENGUAJE MÁQUINA

```
0000:  01 11 00 00 00 EE 63 6C 61 73 69 66 69 63 61 63
0010:  69 6F 6E 04 08 00 00 EE 73 65 63 74 31 0F 00 10
0020:  00 00 21 00 00 21 E8 2D 41 59 20 41 53 4D 2D 43
0030:  4C 41 53 20 0A 0D 00 43 6F 6D 69 65 6E 7A 6F 20
0040:  0A 0D 00 66 69 6E 20 20 0A 0D 00 4E 75 6D 65 72
0050:  6F 20 64 65 20 70 61 72 61 6D 65 74 72 6F 73 20
0060:  6E 6F 20 76 61 6C 69 64 6F 20 0A 0D 00 6E 75 6D
0070:  65 72 6F 20 64 65 20 65 6C 65 6D 65 6E 74 6F 73
0080:  3A 20 20 00 10 00 5E 21 14 0C 25 00 00 10 41 0C
0090:  21 7F 59 97 E1 0B 01 00 02 EE 30 95 E6 21 63 21
00A0:  02 02 00 95 E4 21 46 14 0C 25 00 00 46 41 0F 21
00B0:  7F 59 A1 6C 21 0D 00 06 7F 5E 7F 5A 81 66 21 0A
00C0:  00 00 AB 61 0B 06 00 02 E1 24 21 07 00 00 22 71
00D0:  28 07 00 21 A9 71 22 71 29 07 00 41 0D 21 8B 98
00E0:  EA 07 A1 98 8B 67 EE F8 0B 0A 00 00 E6 15 E8 EA
00F0:  21 0A 00 01 AD 98 AB 71 22 73 28 07 00 21 A9 71
0100:  22 73 29 07 00 21 E8 EF 21 14 0C 25 00 00 24 41
0110:  08 21 7F 59 0B 01 00 00 E6 03 95 EA F1 82 9E 08
0120:  14 0C 25 00 00 1C 41 03 21 7F 59 7F 5A 9E 08 03
```

PROGRAMA ESCRITO EN LENGUAJE COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EJEMPLO.
DATE-WRITTEN. OCT-85.
.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. L1.
OBJECT-COMPUTER. L1.
INPUT-OUTPUT SECTION.
       FILE CONTROL.
       SELECT FICH-MOV
       ASSIGN TO DISK "MOVIMIENTOS"
       ORGANIZATION IS SEQUENTIAL
       ACCESS MODE IS SEQUENTIAL


DATA DIVISION.
FILE SECTION.
       FD FICH MOV
       LABEL RECORDS ARE OMITTED
       DATA RECORD IS REC-MOV.

```

```
          01  REC-MOV.
              02  CLIENT   PIC X(20).
              02  CODCLI   PIC 9(5).
              02  DIREC    PIC X(20).
              02  ARTIC    PIC X(12).
              02  DESC     PIC X(35).
              02  CODART   PIC X(5).
              02  COST     PIC 9(6).
              02  CORREC   PIC X(8).
WORKING-STORAGE SECTION.

77        C-TIP-REC        PIC X(3)


SCREEN SECTION.
          01  VIDEO BLANK SCREEN
              02  LINE 4 COLUMN 4 VALUE "CLIENTE".
              02  LINE 4 COLUMN 26 VALUE "COD.CLIENTE".
              02  LINE 4 COLUMN 48 VALUE "DIRECCION".
              02  LINE 8 COLUMN 4 VALUE "ARTICULO".
              02  LINE 8 COLUMN 26 VALUE "COD.ARTICULO".
              02  LINE 8 COLUMN 48 VALUE "DESCRIPCION".
              02  LINE 8 COLUMN 65 VALUE "COSTE".

          01  FIN-DE-TRABAJO
              02  FILQ BLANK SCREEN.
          01  ERROR
              02  FILR BELL.

          01  VALUES.
              02  C-CLIENT LINE 4 COLUMN 4      PIC X(20) USING CLIENT.
              02  C-CODCLI LINE 4 COLUMN 24     PIC 9(5) USING CODCLI.
              02  C-DIREC LINE 4 COLUMN 44      PIC X(20) USING DIREC.
              02  C-ARTIC LINE 8 COLUMN 4       PIC X(12) USING ARTIC.
              02  C-DESC LINE 8 COLUMN 24       PIC X(35) USING DESC.
              02  C-COD-ART LINE 8 COLUMN 44 PIC X(5) USING CODART.
              02  C-COST LINE 8 COLUMN 65       PIC 9(6) USING COST.


PROCEDURE DIVISION.
PRINCIPIO.
          DISPLAY VIDEO.
          OPEN I-O FICH-MOV
CICLO-DE-TRABAJO.
          PERFORM ACERAR-CAMPOS
          READ FICH-MOV
                AT END GO TO FIN-PROC.
          DISPLAY VALUES.
          ACCEPT C-TIP-REC.
          IF C-TIP-REC = "S"
          PERFORM TRANSACCION1 THRU FIN-TRANSACCION2
                    ELSE
          PERFORM TRANSACCION2 THRU FIN-TRANSACCION2.
          GO TO CICLO DE TRABAJO.
FIN-PROC.
          CLOSE FICH-MOV.
          STOP RUN.

ACERAR CAMPOS.
          MOVE ZERO TO CLIENT, CODCLI, DIREC, ARTIC, CODART, COST, CORREC.
          MOVE SPACES TO C-TIP-REC.
```

```
TRANSACCION1.
        MOVE "CORRECTO" TO CORREC.
        REWRITE REC-MOV.
FIN-TRANSACCION1.

TRANSACCION2.
        MOVE "NO CORRECTO" TO CORREC.
        REWRITE RECMOV.
FIN-TRANSACCION2.
```

PROGRAMA ESCRITO EN LENGUAJE FORTRAN

```
C   CÁLCULO DEL ÁREA DE UN TRAPECIO
C
C
    READ (1,10) NUM, BASE1, BASE2, ALTURA
10  FORMAT (I5, 1X, I5, 1X, I5, 1X, I5)
    ÁREA = ((BASE1+BASE2)/2)*ALTURA
    WRITE (2,20) NUM,AREA
20  FORMAT (1X, 'EL ÁREA DEL TRAPECIO NÚMERO  ', I5, ' ES ', I5)
```

Resultado del programa	
	EL ÁREA DEL TRAPECIO NÚMERO 1 ES 8
	EL ÁREA DEL TRAPECIO NÚMERO 2 ES 12
	EL ÁREA DEL TRAPECIO NÚMERO 3 ES 28

PROGRAMA ESCRITO EN LENGUAJE PASCAL

```
PROGRAM  JUEVES (INPUT, OUTPUT);

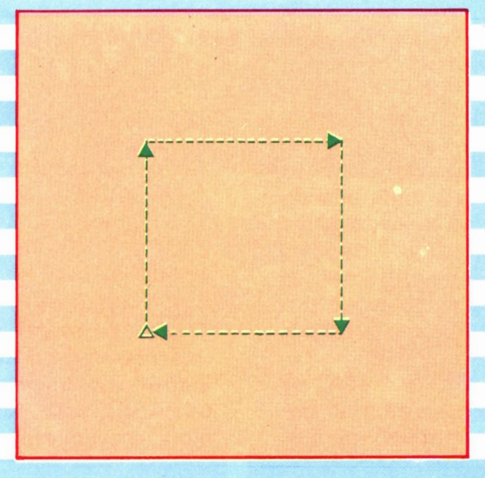
LABEL 1;
CONST  DIAS-SEMANA=7;
TYPE   KEYBOARD-INPUT=CHAR;
VAR    KEYIN : KEYBOARD-INPUT;

BEGIN
     WRITE ('HOY, ¿ES JUEVES?');
1:   READLN (KEYIN);
     CASE KEYIN OF
     'S', 's' : WRITELN ('es jueves.');
     'N', 'n' : WRITELN ('no es jueves.');
   OTHERWISE
     WRITELN ('Digitar S o N.');
     WRITE ('Por favor vuelva a digitar:');
       GOTO 1
   END
```

PROGRAMAS LOGO EN LOS QUE SE REALIZAN FIGURAS GEOMÉTRICAS CON LA TORTUGA

Para que la TORTUGA dibuje un cuadrado se podría construir el siguiente programa LOGO:

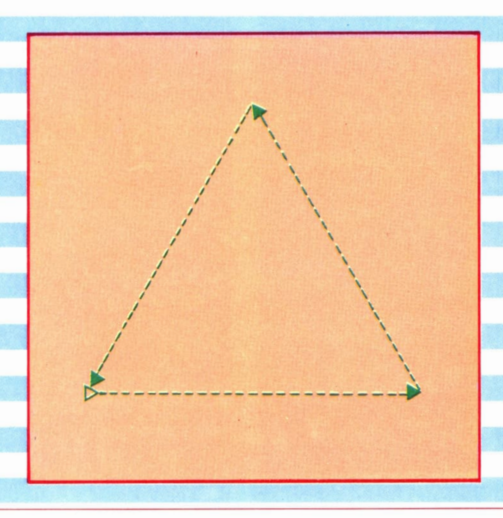
```
CS       borrado de la pantalla
FD 30    indica a la tortuga que desde donde está
         avance 30 unidades en la dirección que señale extremo
RT 90    indica a la tortuga que varíe su posición 90° a la derecha
FD 30    le indica que avance 30 unidades
TR 90    le indica que varíe su posición 90° a la derecha
FD 30    y así sucesivamente hasta formar el cuadrado tal como
         se indica en la figura
RT 90
FD 30
END
```



Veamos ahora cómo puede construirse un triángulo:

```
CS       borrado de pantalla
FD 30    avance de 30 unidades
LT 120   variación de 120° a la izquierda
FD 30    avance de 30 unidades
LT 120   variación de 120° a la izquierda
FD 30    avance de 30 unidades
LT 120   variación de 120° a la izquierda para quedar en su posición inicial
END
```

Con este programa podemos construir el triángulo de la figura.



El lenguaje BASIC

El nombre de BASIC responde a las siglas de *Beginner's All Symbolic Instruction Code,* o sea, código de instrucciones simbólicas de uso general para principiantes.

Este lenguaje fue desarrollado, junto con el primer sistema de tiempo compartido del mundo, en un colegio americano para servir de lenguaje de aprendizaje a los alumnos. Sus autores fueron John G. Kemeny y Thomas E. Kurts. En tal colegio, los estudiantes podían comunicarse con la computadora gracias a que cada uno de ellos tenía un terminal conectado a ella; gracias al BASIC se podía codificar, ejecutar y depurar un programa en muy poco tiempo. Además, al ser muy fácil de aprender, los estudiantes pudieron empezar a escribir programas tras seguir un curso de muy pocas horas.

Años después de su creación, el BASIC se ha convertido en uno de los lenguajes más populares y fáciles de usar. Se han diseñado lenguajes BASIC para toda clase de computadoras, desde las de bolsillo hasta las grandes computadoras.

Su característica principal consiste en que es muy fácil de usar por los principiantes en programación. Sin embargo, veamos las principales características que lo hacen diferente a los demás lenguajes.

• *Se puede dar un nombre muy corto a las variables del BASIC.* Las variables pueden ser numéricas (A = 1) y alfanuméricas (A$ = "PEPE"). Los nombres de tales variables pueden empezar por cualquier letra del alfabeto.

• *El BASIC permite programar de una forma conversacional.* Gracias a que es un lenguaje intérprete, es decir que no necesita compilación (ya que cada instrucción se traduce inmediatamente antes de ejecutarse), se puede cargar un programa escrito en BASIC en memoria, ejecutarlo, corregirlo directamente si tiene algún error y volverlo a salvar en el dispositivo donde se tuviera almacenado.

• *Los mensajes de error generados por el BASIC son explícitos y numerosos.* En un lenguaje de programación, la cantidad de mensajes de error que éste puede generar y la calidad de estos mensajes constituyen una característica importante de cara a poder medir la potencia de tal lenguaje de programación. Los mensajes de error son muy útiles para el programador, ya que le indican, en el momento de producirse, el tipo de error de que se trata y a veces el porqué se ha producido. En el caso del BASIC, la información que el lenguaje da cuando se produce un error es muy completa; incluso, en caso de que se produzca un error en un programa BASIC, éste, además del mensaje de error, indica también la línea de programa donde se ha producido tal error.

El BASIC que explicaremos en este capítulo será fundamentalmente el BASIC que se utiliza en la mayoría de computadoras personales, en las cuales este lenguaje alcanza toda su potencia y donde se le saca mayor rendimiento.

Sin embargo, en este capítulo solamente describiremos detalladamente cada una de las instrucciones principales o más comunes de este BASIC; pero, antes de empezar esta descripción, explicaremos la «sintaxis» que vamos a utilizar para realizar tal descripción y las convenciones en esta sintaxis.

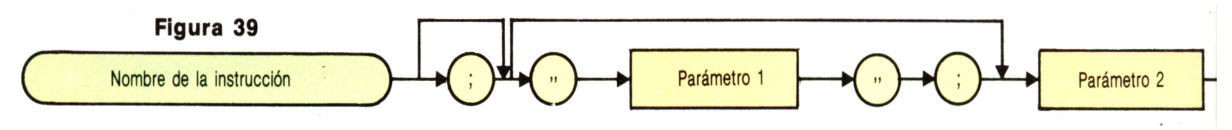
CONVENCIONES EN LA NOTACIÓN

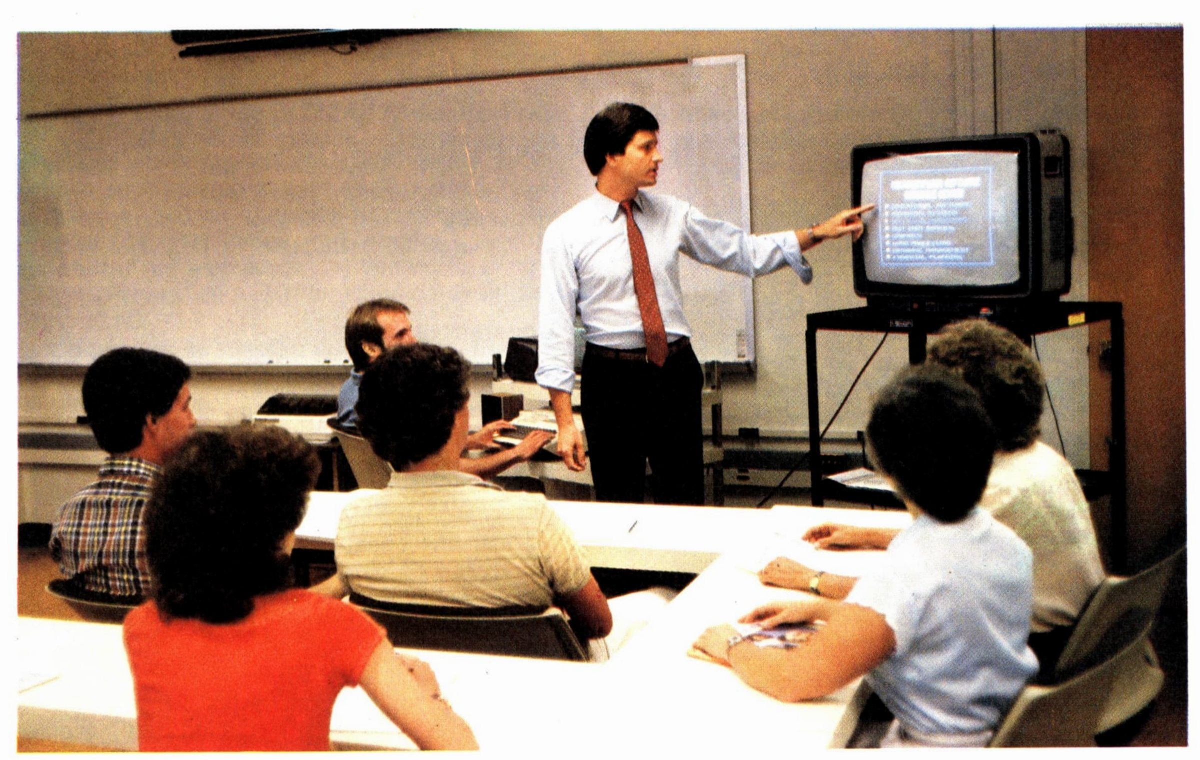
Para describir cada una de las instrucciones BASIC, primero indicaremos en un diagrama el nombre de la instrucción y los parámetros que puede llevar dicha instrucción, de tal manera que el nombre de la instrucción irá dentro de un rectángulo. Los signos especiales se indicarán dentro de círculos y, cuando un parámetro no sea obligatorio, el enlace entre un parámetro y otro o entre el nombre de instrucción y sus parámetros puede ir directamente de parámetro en parámetro o saltarse el parámetro o parámetros no obligatorios.

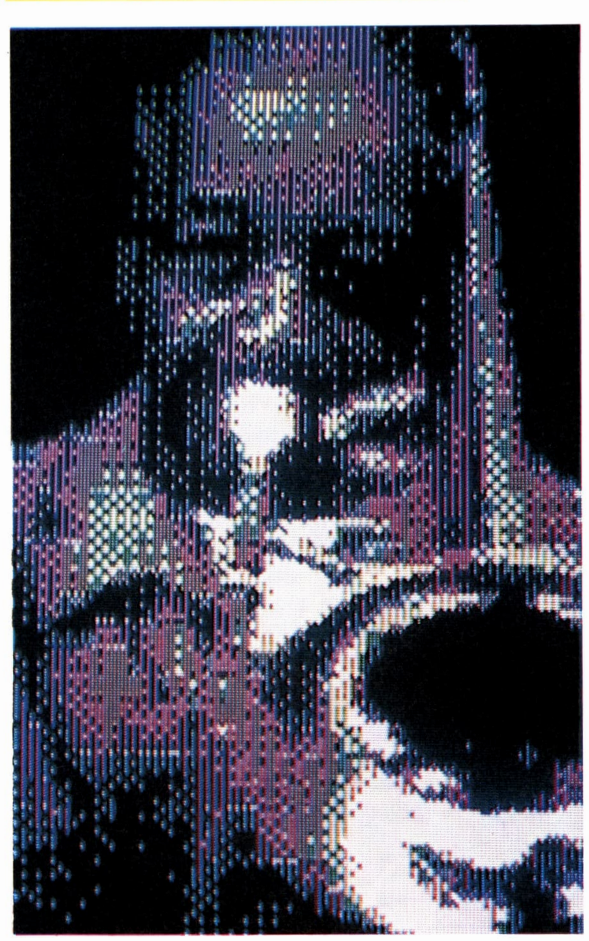
Para comprenderlo mejor veamos uno de estos diagramas **(figura 39)**.

Arriba, enseñanza de computación en lenguaje BASIC. Las grandes facilidades del BASIC lo han convertido en el lenguaje de programación por excelencia para el aprendizaje de principiantes, ya que permite programar sin hacer uso de compiladores, sus mensajes de error son explícitos y numerosos y admite nombres muy cortos para sus variables. Sin embargo, existe un lenguaje BASIC experto, el BASIC PLUS que permite la realización de gráficos. Abajo, gráficos realizados mediante Diseño Asistido por Computadora.

Figura 39







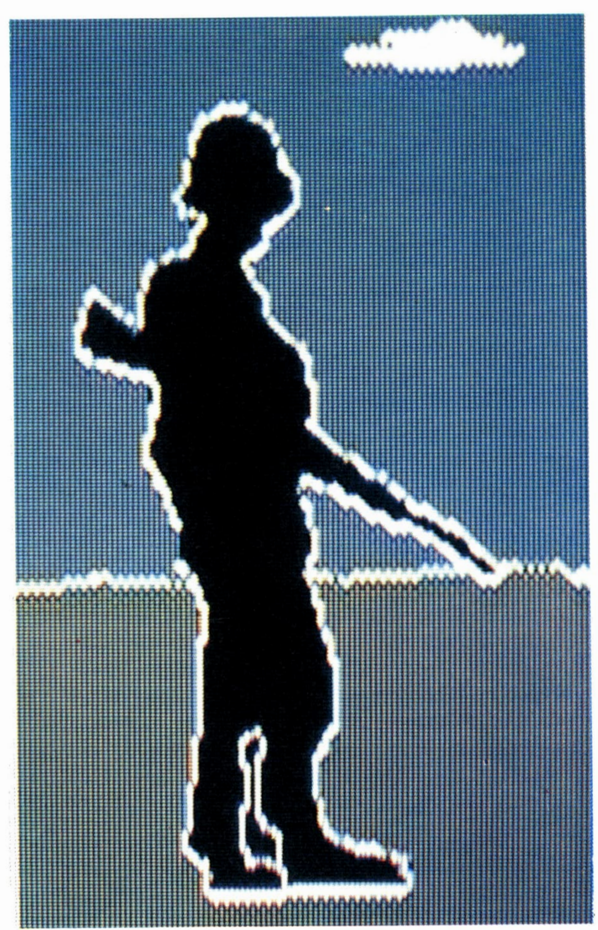


Figura 39 a

Nombre de la instrucción → ; → " → Parámetro 1 → " → ; → Parámetro 2 →

En este diagrama se está describiendo una instrucción en la cual existen parámetros y, por lo tanto, tal instrucción podrá tener varios formatos.

El primer formato está constituido por todo el diagrama y sería el que está representado en la **figura 39a**.

El segundo formato sería el que aparece en la **figura 39b**.

Como vemos, en este formato no se especifica el punto y coma (;), porque es opcional, tal como se ve en el primer diagrama.

El tercer formato es el de la **figura 39c**.

En este formato se especifica el nombre de instrucción, el (;) y el último parámetro.

El cuarto formato es el correspondiente a la **figura 39d**.

Como vemos, este formato es el más corto y solamente está formado por el nombre de la instrucción y el último parámetro.

Veamos cada uno de los formatos de esta instrucción en un caso real:

```
LINE INPUT ; "Nombre :" ; N$      LINE INPUT — Nombre instrucción
LINE INPUT "Nombre :" ; N$        Nombre: —— Parámetro 1
LINE INPUT ; N$                   N$ ———— Parámetro 2
```

ELEMENTOS DE UNA INSTRUCCIÓN BASIC

Cada instrucción BASIC, como hemos visto en el apartado anterior, consta del nombre de la instrucción, de unos parámetros y de unos signos especiales. El nombre de la instrucción se corresponde con una palabra reservada o clave, los parámetros pueden ser constantes, variables o expresiones.

Palabras clave

En BASIC, cada instrucción empieza con una palabra clave, que es una palabra reservada y que corresponde a la lengua inglesa. Desde el punto de vista sintáctico, siempre debe ser precedida y seguida, al menos, de un espacio. Se las llama reservadas, porque tales palabras no pueden usarse para otra cosa que para especificar el nombre de una instrucción; es decir, no pueden usarse, por ejemplo, como nombres de variables. Supongamos la palabra clave INPUT; tal palabra se corresponde con el nombre de una instrucción BASIC y, por lo tanto, no puede usarse como un nombre de variable.

Algunas veces una instrucción contiene más de una palabra clave; por ejemplo, la instrucción IF.....THEN contiene las palabras IF, al principio de la instrucción, y THEN, al final.

Constantes

Dentro de un programa BASIC podemos encontrarnos con dos tipos de constantes:

— las numéricas;
— las alfanuméricas.

Cualquier cantidad numérica expresada en una instrucción BASIC será una constante numérica; una constante alfanumérica será cualquier serie de caracteres expresados entre comillas, salvo en algunas instrucciones en las que no van entre comillas por el propio formato de tales instrucciones, aunque igualmente son series de caracteres alfanuméricos.

El valor de las constantes siempre será el mismo a lo largo de la ejecución de un programa BASIC.

Las constantes numéricas pueden ser positivas o negativas y enteras o reales, ejemplo 200; −200; 201,50; −201,50.

Variables

Las variables dentro de un programa sirven para representar un dato mediante un nombre. Dicho dato puede variar a lo largo de la ejecución de un programa; por lo tanto, podemos decir que el valor de una variable puede cambiar a lo largo de la ejecución de un programa.

La longitud del nombre de una variable depende del BASIC que estamos utilizando; es decir, cada computadora puede utilizar diferentes lenguajes BASIC con diferentes especificaciones, siendo ésta una de ellas. Los diferentes tipos de BASIC tienen una parte común (la mayor) y la otra con especificaciones propias de cada uno.

Tipos de variables

Las variables se dividen en dos tipos:

— variables simples;
— variables con índice.

Las *variables simples* son variables cuyo nombre solamente representa un dato. En cambio, las *variables con índice* representan el ele-

Figura 39 b

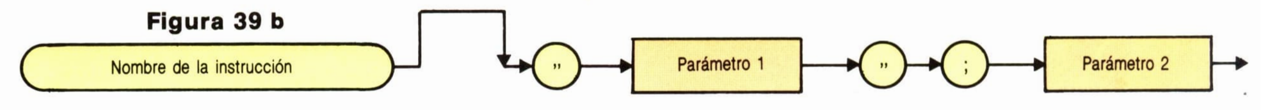


Figura 39 c

mento de una matriz o vector; el nombre de la variable es el mismo para todos los elementos del vector y, para distinguir un elemento de otro, se utiliza el índice, que variará desde 1 hasta el n.º de elementos del vector.

Un nombre de variable simple puede ser PRODUCTO.

Un nombre de variable con índice podría ser SUMA (1), que representaría al primer elemento del vector SUMA.

25	50	100	75	400	300
SUMA (1)	SUMA (2)	SUMA (3)	SUMA (4)	SUMA (5)	SUMA (6)

En este caso, SUMA (1) = 25, SUMA (2) = 50, SUMA (3) = 100, SUMA (4) = 75, SUMA (5) = 400 y SUMA (6) = 300.

De esta manera se pueden representar elementos de vectores y de matrices de cualquier dimensión. En el caso de una matriz de tres dimensiones, cada elemento que la compone se puede representar por tres índices, uno por cada dimensión. Por ejemplo, SUMA (1, 2, 5) sería el nombre de la variable que representaría a un elemento de la matriz SUMA de tres dimensiones.

Tanto las variables simples como las variables con índice pueden ser:

— numéricas;
— alfanuméricas.

El nombre de las *variables numéricas* puede estar formado de cualquier combinación de caracteres, siempre que su primer carácter sea alfabético. Sin embargo, según el carácter con el que terminen, pueden ser de tres tipos:

— enteras;
— de simple precisión;
— de doble precisión.

• *Las variables enteras* son variables cuyo valor puede variar entre −32768 y 32767 y cuyo nombre termina con el signo especial %. Por ejemplo, el nombre AREA% sería el nombre de una variable de este tipo.

• *Las variables de simple precisión* son variables numéricas cuyo valor puede variar entre $\pm 10^{-38}$ y $\pm 10^{38}$, y cuyo nombre termina con el signo especial !. Por ejemplo RESTA! sería el nombre de una variable de este tipo.

• *Las variables de doble precisión* son variables numéricas cuyo valor puede variar entre $\pm 10^{-308}$ y $\pm 10^{308}$ y cuyo nombre termina con el signo especial £ (libra) o el signo #. Por ejemplo, DIV£ o DIV# es un nombre de variable de este tipo.

Normalmente, en el tipo de BASIC que estamos viendo, cuando una variable no lleva signo especial al final de su nombre, ésta será una variable numérica de simple precisión.

Las *variables alfanuméricas* forzosamente deben terminar con el signo especial $ (dólar).

El valor de tales variables es siempre una serie de caracteres alfanuméricos cuyo límite generalmente es de 255.

Para comprender mejor las divisiones de las variables, podemos construir un cuadro como el siguiente:

Tipos de variables	
Simples	***Con índice***
Numéricas — Enteras — Simple precisión — Doble precisión Alfanuméricas	Numéricas — Enteras — Simple precisión —Doble precisión Alfanuméricas

Expresiones

Las expresiones pueden dividirse en 4 categorías:

— expresiones numéricas;
— expresiones alfanuméricas;
— expresiones de comparación;
— expresiones lógicas.

EXPRESIONES NUMÉRICAS

Una expresión numérica no es sino una serie de variables y constantes con las que se realiza un cálculo mediante operadores aritméticos. En BASIC existe una instrucción que permite asignar a una variable el resultado de una expresión aritmética. El nombre de esta instrucción es LET, y con ella podremos ilustrar un ejemplo de expresión aritmética:

```
LET A=(50*N)/2
```

En este ejemplo vemos que a la variable A se le asigna el resultado de la expresión aritmética de la derecha del signo igual, en la que vemos variables (N), constantes (50, 2) y operadores aritméticos (*, /).

Figura 39 d

Los operadores aritméticos que se pueden utilizar en cualquier expresión aritmética del BASIC y su significado son los que aparecen en el cuadro **Operadores aritméticos**.

La prioridad de estos operadores aritméticos en una expresión numérica en la que no haya paréntesis, es decir, el orden en que se efectuarían las operaciones en una expresión numérica que no tuviera paréntesis y en la que hubiera todos los operadores antes indicados, sería:

Primero la potenciación, después la multiplicación y división con el mismo orden, seguidamente la división entera, a continuación el cálculo del resto de la división y por último la suma y la resta con el mismo orden. Cuando dentro de una expresión numérica hay paréntesis, siempre tiene mayor prioridad lo que va dentro de los paréntesis.

Por ejemplo, el resultado de la expresión 5+4*2 será 8+5=13, porque se efectúa primero la multiplicación y después la suma. Sin embargo, el resultado de la expresión (5+4)*2 será 9*2 = 18, porque se ha efectuado primero la suma y después la multiplicación. Por lo tanto, vemos que una misma expresión puede tener un resultado diferente según se coloquen paréntesis o no. Notemos también que la expresión 5+(4*2) es la misma que 5+4*2.

EXPRESIONES ALFANUMÉRICAS

En el lenguaje BASIC es posible realizar operaciones con constantes y variables alfanuméricas para formar expresiones alfanuméricas. Tales operaciones, sin embargo, se reducen a una sola, que se llama concatenación y cuyo operador es el signo de la suma +.

Por ejemplo, si tenemos una variable alfanumérica cuyo nombre es A$ y cuyo valor es ''ALBERT'' y otra cuyo nombre es B$ y su valor ''EINSTEIN'', al realizar el cálculo de la expresión alfanumérica A$+B$ y asignar el resultado a otra variable alfanumérica cuyo nombre sea C$, de modo que la instrucción BASIC sería LET C$ = A$+B$, entonces en C$ obtendríamos ''ALBERT EINSTEIN'', es decir, C$ = = A$+''EINSTEIN'', en la que C$ volvería a tener valor ''ALBERT EINSTEIN''.

EXPRESIONES DE COMPARACIÓN

Una expresión de comparación, como su nombre indica, compara dos o más expresiones numéricas o alfanuméricas mediante operadores de comparación. En este tipo de operaciones no está permitido comparar expresiones de distinto tipo, es decir, expresiones numéricas con alfanuméricas y al revés.

Operadores aritméticos

SÍMBOLO	OPERACIÓN	EJEMPLO
+	Suma	Si la variable A=2, entonces, en la expresión B=A+5, en B se almacenará 2+5, es decir 7
—	Resta	Si la variable A=2, entonces, en la expresión B=10—A, en B se almacenará 10—2, es decir 8
\	División entera. Los operadores se truncan al entero más próximo y, si el resultado da decimales, también éste se trunca al número entero más próximo.	El resultado de la expresión 13\4 sería 3, aunque en realidad sería 3,25, pero como si hay decimales se trunca, se queda en 3. El resultado de la expresión 16,7\3,4 sería 6, porque 16,7 pasa a ser 17 y 3,4 pasa a ser 3. La expresión queda 17\3 y el resultado de la división es 5,66, que al truncarlo será 6, porque está más cerca de 6 que de 5.
MOD	Cálculo del resto de una división	El resultado de la expresión numérica 10 MOD 4 será 2, porque al dividir 10 entre 4 nos queda de resto 2. Así, en la instrucción LET B=10 MOD 4, B tomará el valor 2
*	Multiplicación	El resultado de la expresión numérica 5 * 4 será 20
/	División	El resultado de la expresión numérica 22/6 será 3,66666
^	Potenciación	El resultado de la expresión numérica 4 ^ 2 será 16, porque $4^2=4\times4=16$

Operadores lógicos								
A	**NOTA**	**A**	**B**	**A AND B**	**A OR B**	**A XOR B**	**A EQV B**	**A IMP B**
1	Ø	1	1	1	1	Ø	1	1
Ø	1	1	Ø	Ø	1	1	Ø	Ø
		Ø	1	Ø	1	1	Ø	1
		Ø	Ø	Ø	Ø	Ø	1	1

Los operadores que se utilizan en este tipo de expresiones son los siguientes:

Signo	Significado
=	Comparación por igual
>	Comparación por mayor
<	Comparación por menor
> = o = >	Comparación por mayor o igual
< = o = <	Comparación por menor o igual
< > o > <	Comparación por no igual o diferente

Ejemplos de expresiones de comparación:

— A*B > C+D;
— A$+B$ > C$;
— A / C = A+B;
— A+B>C$ no correcta, ya que se comparan dos expresiones de distinto tipo.

La manera de realizar la comparación es como sigue. Primero se calcula el resultado de la expresión numérica o alfanumérica de la izquierda del operador, después se calcula a su vez el resultado de la expresión numérica o alfanumérica de la derecha del operador y, una vez calculados estos resultados, se comparan de acuerdo con el operador de comparación que se encuentra entre las dos expresiones.

Este tipo de expresiones también se llama expresiones de relación y a sus operadores, operadores de relación.

EXPRESIONES LÓGICAS

Las expresiones lógicas sirven para enlazar varias expresiones de otro tipo, numéricas o de comparación, mediante un operador lógico.

Por ejemplo, si especificamos en un programa la instrucción BASIC siguiente: IF A>B AND C>D, estamos enlazando la expresión de comparación A>B con la del mismo tipo C>D. Como vemos, la instrucción tiene como palabra clave IF, que sirve para comprobar si se cumple una determinada expresión o no. En este caso, la expresión sería verdadera si el valor de la variable A fuera mayor oue el de la variable B y el de la variable C mayor que el de la D. Si una de estas dos condiciones no se cumpliera, tampoco se cumpliría la expresión total.

Los operadores lógicos más comunes que podemos encontrar son los descritos en la tabla **Operadores lógicos**, en la que también se describe su actuación.

Si suponemos que un 1 es verdadero y que un Ø es falso, podremos interpretar la tabla antes mencionada. En ella los principales operadores lógicos son:

NOT, AND, OR, XOR, IMP, EQV

Estos operadores, al igual que los operadores numéricos, se ejecutan según un nivel de prioridades, que es el expuesto más arriba; es decir, primero se ejecutará el operador NOT, después AND, seguidamente OR, después XOR, a continuación IMP y por último EQV.

Uso del carácter «ESPACIO» en las instrucciones BASIC

Normalmente, dentro de las instrucciones BASIC, y por tanto en las líneas de programa, se acostumbra a utilizar el carácter «espacio» para hacer más legible el programa a cualquier persona que pueda leerlo, o incluso para la misma persona que lo construye. Lógicamente, en tal caso existe alguna limitación y alguna excepción.

- Dentro de una instrucción BASIC, la palabra clave debe estar precedida por un espacio y también estar seguida por él.
- El carácter «espacio» sólo es significativo en el interior de una constante alfanumérica.
- No se admite tal carácter dentro de una constante numérica, dentro de una palabra clave o nombre de instrucción y dentro del nombre de una variable. Por ejemplo:

```
10 INPUT ''Área''; A es lo mismo que
10 INPUT ''Área''; A,
```

y sin embargo, es diferente a

```
10 INPUT ''Á r e a''; A.
```

COMENTARIOS

El BASIC, al igual que otros lenguajes de programación, permite documentar los programas que con él se construyen. Esto puede hacerse mediante la instrucción REM, que trata todos los caracteres que tenga a continuación, como comentario del programa.

Por ejemplo:

10 REM Programa ejemplo
: :
: :

100 REM Cálculo de la media aritmética de A, B y C

Estas dos instrucciones indican que, a partir de la instrucción número diez, empieza un programa cuyo nombre es «ejemplo» y que, a partir de la instrucción 100 de este programa es donde se va a calcular la media aritmética de tres variables A, B y C.

INTRODUCCIÓN, LISTADO, EJECUCIÓN Y ALMACENAMIENTO DE UN PROGRAMA BASIC

Antes de empezar a describir las instrucciones más importantes del lenguaje BASIC, explicaremos cómo se realiza la introducción de un programa BASIC en la computadora, cómo se efectúa un listado de tal programa por pantalla y cómo se hace para ejecutar dicho programa.

Introducción de un programa

La introducción de un programa BASIC en una computadora se hace mediante el teclado y digitando instrucción por instrucción.

En cada línea de programa se introducirá el n.º de la línea y después la instrucción o instrucciones que conformarán esa línea. Para indicarle a la computadora el final de una línea de programa, siempre se pulsa una tecla especial que poseen todos los teclados que sirve para indicar fin de entrada de un dato y puede llamarse «ENTER», «CR», «ACCEPT», etc. Sin embargo, en nuestro caso la llamaremos «CR».

Veamos un ejemplo de la introducción de un programa:

```
10 REM CÁLCULO DEL ÁREA DE UN
   CUADRADO [CR]
20 INPUT "Lado"; A [CR]
30 IF A<=0 THEN GOTO 20 [CR]
40 LET ÁREA=A*A [CR]
50 PRINT "Área="; ÁREA; "Lado=";A [CR]
60 END [CR]
```

Fijémonos que al final de cada línea de programa aparece un cuadrado con un CR en su interior. Esto significa que hay que pulsar CR después del último carácter de la línea.

El programa aparecerá en la pantalla de la computadora tal como se muestra aquí, pero sin los cuadros con CR.

Obsérvese que el número de cada línea de programa salta de 10 en 10; sin embargo, no tiene por qué ser así, sino que puede numerarse como uno desee siempre que se guarde un orden. No obstante, se acostumbra a hacer así en prevención de que si más tarde se quiere introducir una línea entre dos que ya estén hechas, ésta pueda tener un número de línea comprendido entre el de la línea anterior y el de la posterior. Por ejemplo, si quisiéramos introducir una línea entre las líneas 30 y 40 del programa anterior, no tendríamos ningún problema, ya que podríamos darle del n.º 31 al 39.

Antes de seguir es necesario explicar que en lenguaje BASIC existen dos tipos de instrucciones: unas sirven para realizar tareas de manejo de programas que no pueden formar parte de ellos, las cuales se llaman inmediatas; otras, las más numerosas, pueden ser inmediatas y a la vez formar parte de un programa.

Para poder explicar cómo se trabaja con un programa BASIC, se han de explicar al mismo tiempo las instrucciones que se utilizan para realizar este trabajo.

Como primer paso destaquemos la existencia de una instrucción inmediata que permite ir numerando automáticamente cada línea de instrucción, con lo cual se evita que tenga que introducir los números el programador. El formato de tal instrucción es el descrito en la **figura 40**, que se interpreta de la siguiente manera:

AUTO: Nombre de la instrucción.

N.º de línea: Número por el que se quiere empezar a numerar las líneas. Si se omite su valor será 0 si también se omite el parámetro «intervalo»; si este último no se omite, su valor será 10.

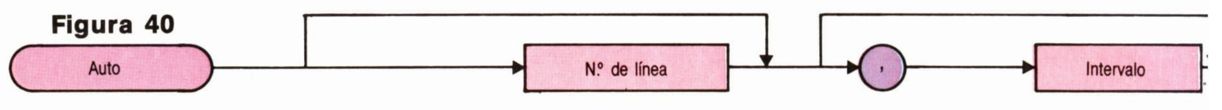
INTERVALO: Cantidad de diferencia entre un número de línea y otro. Si se omite, su valor será siempre 10.

EJEMPLOS DE TODOS LOS CASOS

AUTO [CR] Empezará a enumerar por la línea 10 y el intervalo será de 10 en 10.

10 ______
20 ______
30 ______

Figura 40



AUTO, 5 CR Empezará a enumerar por la línea Ø y el intervalo será de 5 en 5.

Ø ______
5 ______
10 ______

AUTO 20 CR Empezará a enumerar por la línea 20 y el intervalo será de 10 en 10.

20 ______
30 ______
40 ______

AUTO 20,20 CR Empezará a enumerar por la línea 20 y el intervalo será de 20 en 20.

20 ______
40 ______
60 ______

Una vez se ha terminado la introducción del programa, para indicarle a la computadora que no genere más números de línea, normalmente se hace pulsando las teclas CTRL C simultáneamente.

Listado de un programa

El programa introducido queda almacenado en la memoria central de la computadora, de tal manera que, si borramos la pantalla, todavía tenemos la posibilidad de poder listarlo de nuevo en ella o, si se quiere, por impresora. Esto se consigue mediante otras instrucciones inmediatas cuyo nombre es LIST y LLIST.

El formato de tales instrucciones es el descrito en la **figura 41** y se interpreta así:

LIST y LLIST: Nombres de las instrucciones. La única diferencia entre ellas consiste en que una actúa sobre la pantalla (LIST) y la otra sobre la impresora (LLIST).

N.º de línea 1: Número de línea del programa a partir de la cual se quiere visualizar en pantalla o listar en impresora el programa. Si se omite, se empezará a visualizar o listar desde la primera línea del programa.

N.º de línea 2: Número de línea del programa hasta la que se quiere visualizar o listar el programa. Si se omite, se listará o visualizará hasta el final del programa.



Técnico utilizando un tablero digitalizador, que permite convertir una imagen en bits: 0 y 1.

EJEMPLOS DE TODOS LOS CASOS

LIST CR	Visualizará en pantalla todo el programa desde la primera línea hasta la última.
LLIST CR	Hará lo mismo que el ejemplo anterior pero en impresora.
LIST 200 CR	Se visualizará sólo la línea 200.
LLIST 200 CR	Se imprimirá sólo la línea 200.
LIST 100- CR	Se visualizará el programa a partir de la línea 100 hasta el final.
LLIST 100- CR	Se imprimirá el programa a partir de la línea 100 hasta el final.

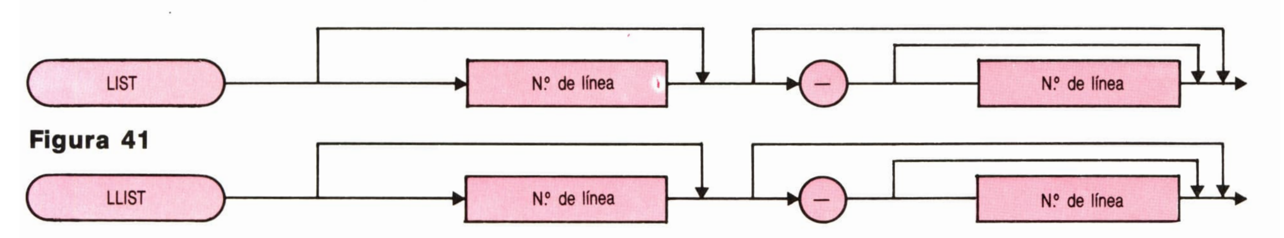


Figura 41

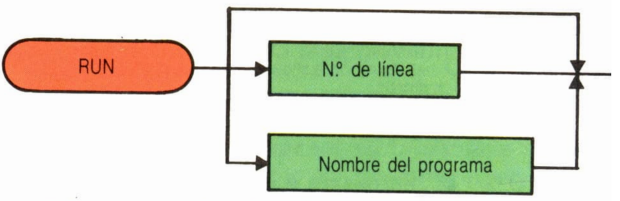
LIST 100-500 [CR]	Se visualizará la porción de programa que va de la línea 100 a la 500.
LLIST 100-500 [CR]	Se imprimirá la porción de programa que va de la línea 100 a la 500.
LIST -500 [CR]	Se visualizará desde la primera línea a la número 500.
LLIST -500 [CR]	Se imprimirá desde la primera línea a la número 500.

El usuario o programador siempre puede interrumpir la visualización o impresión del programa pulsando simultáneamente las teclas [CTRL] [C].

Ejecución de un programa

El programa que hemos introducido, además de visualizarlo en pantalla y listarlo en impresora, se puede ejecutar mediante la instrucción RUN, que puede ser una instrucción inmediata o de programa. Su formato es:

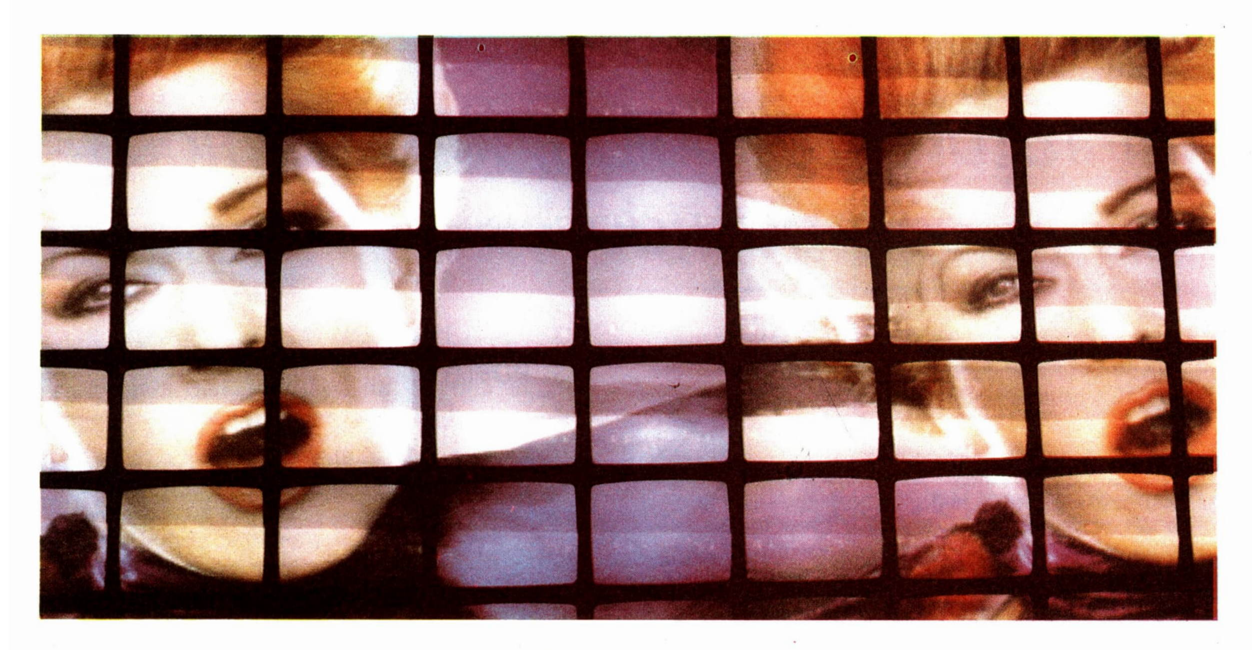
El primer paso para ponerse a trabajar con una computadora es cargar el programa en la máquina. Los programas de utilidad para usuarios se almacenan en un dispositivo de almacenamiento externo: cassette, floppy, etc. En general, la mayoría de los usuarios carecen de la competencia necesaria para escribir programas largos, pero en el mercado existe una amplia gama de programas adaptados a las necesidades de profesionales y empresas.



RUN:	Nombre de la instrucción.
N.º de línea:	N.º de línea a partir de la que se quiere ejecutar el programa. Si se omite, toma el valor de la 1.ª línea del programa.
Nombre del programa:	Nombre del programa BASIC que se quiere ejecutar. Cuando se especifica el nombre del programa, significa que éste no se encuentra en memoria; lo que hace esta instrucción es leerlo de algún dispositivo de almacenamiento externo de memoria, cargarlo en memoria central y ejecutarlo.

EJEMPLOS DE TODOS LOS CASOS

RUN [CR]	Se ejecuta el programa que se encuentra en memoria empezando por la 1.ª línea.
RUN 200 [CR]	Se ejecuta el programa que se encuentra en memoria empezando por la línea 200; es decir, las lí-

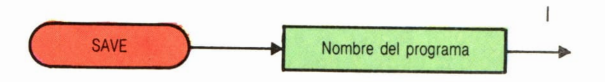


neas de programa comprendidas entre la 1.ª y la 200 no se ejecutarán.

RUN ''CÁLCULO'' CR Se buscará el programa llamado CÁLCULO en un dispositivo de almacenamiento externo de memoria (floppy, cassette, disco, etc.) y, una vez encontrado, se cargará en memoria e inmediatamente después se ejecutará empezando desde la primera instrucción.

Almacenamiento de un programa

Cualquier programa que se encuentre en memoria central puede ser almacenado en un dispositivo de almacenamiento externo de memoria mediante la instrucción SAVE, que puede ser inmediata o de programa. Su formato es el siguiente:

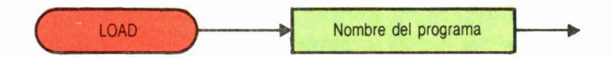


SAVE: Nombre del programa.
Nombre del programa: Nombre del programa que se encuentra en memoria y que se va a almacenar en un dispositivo de almacenamiento externo de memoria.

Una hermosa imagen aparece construida en una batería de pantallas de televisión. El efecto se obtuvo asociando las terminales de vídeo a una computadora con gran capacidad para gráficos.

EJEMPLOS
SAVE ''CALCULO'': Se almacenará en disco, floppy, etc., el programa que se encuentra en memoria central y con el nombre CÁLCULO.

Si un programa BASIC puede almacenarse en un floppy, cassette, etc., lógicamente también debe poderse leer de estos dispositivos, para cargarlo en memoria. Esta función la realiza la instrucción LOAD, que también puede ser inmediata o de programa. Su formato es el siguiente:



LOAD: Nombre de la instrucción.
Nombre del programa: Nombre con el que tal programa fue almacenado por la instrucción SAVE y que se leerá del dispositivo donde esté almacenado y se cargará en memoria central, listo para su ejecución.

EJEMPLO
LOAD ''CÁLCULO'': El programa que se almacenó bajo el nombre de CÁLCULO, se leerá del dispositivo y se cargará en memoria central.

El futuro tecnológico-científico y las computadoras

Los avances científicos más espectaculares se están produciendo en el campo de las ciencias aplicadas, también llamado *tecnológico*. Estos avances se consiguen gracias a los esfuerzos que realizan las grandes compañías multinacionales y los gobiernos que quieren estar en cabeza de la carrera hacia el futuro tecnológico. En efecto, las grandes multinacionales, más que otras compañías, contratan a los mejores científicos y a los técnicos más especializados para que desarrollen nuevas técnicas y aparatos sofisticados con el fin de conseguir su objetivo fundamental: un mayor beneficio económico para la compañía.

Los gobiernos de las grandes potencias mundiales dedican importantes presupuestos económicos para mantener toda una serie de departamentos —en las universidades sobre todo— dedicados a la investigación técnico-científica. En este caso, el fin de esta inversión presupuestaria no es conseguir grandes beneficios económicos, sino colocarse en un puesto relevante dentro de la lista de las naciones tecnológicamente más avanzadas. Con estas aportaciones también se contribuye al avance de la humanidad en cuestiones como sanidad, nivel de vida, etcétera.

A pesar del interés de los gobiernos por estas cuestiones y de las grandes sumas de dinero invertidas, no se hubieran conseguido los hitos alcanzados de no ser por la aplicación de las computadoras a estas técnicas.

Por todo ello podemos afirmar que las computadoras serán un elemento fundamental en el futuro tecnológico y científico, no sólo por la ayuda que su uso representa a los investigadores, sino también por su propia evolución, que, si ahora es ya espectacular, ¡cómo será en el año 2000!

A modo de ilustración se describen a continuación el estado actual y las innovaciones que se esperan en el futuro de tres campos considerados en cabeza dentro del desarrollo tecnológico y científico actual:

— la robótica;
— el espacio;
— las comunicaciones locales.

LA ROBÓTICA

Este término procede de la palabra *robot*. La robótica es, por lo tanto, la ciencia o rama de la ciencia que se ocupa del estudio, desarrollo y aplicaciones de los robots.

Los robots son dispositivos compuestos de sensores que reciben datos de entrada y que pueden estar conectados a la computadora. Ésta, al recibir la información de entrada, ordena al robot que efectúe una determinada acción. Puede ser que los propios robots dispongan de microprocesadores que reciben el *input* de los sensores y que estos microprocesadores ordenen al robot la ejecución de las acciones para las cuales está concebido. En este último caso, el propio robot es a su vez una computadora.

Al oír la palabra robot, a menudo se produce en nuestra mente la imagen de una máquina con forma humana, con cabeza y extremidades. Esta asociación es fruto de la influencia de la televisión o del cine, cuyos anuncios o películas muestran máquinas con forma humana, llamadas *androides*, que generalmente son pura ficción, ya que o son hombres disfrazados de máquina o, si realmente son máquinas, no efectúan trabajos de los que el hombre se pueda aprovechar.

En la actualidad, los avances tecnológicos y científicos no han permitido todavía construir un robot realmente inteligente, aunque existen esperanzas de que esto sea posible algún día.

Hoy por hoy, una de las finalidades de la construcción de robots es su intervención en los procesos de fabricación. Estos robots, que no tienen forma humana en absoluto, son los encargados de realizar trabajos repetitivos en las cadenas de proceso de fabricación, como por ejemplo: pintar al spray, moldear a inyección, soldar carrocerías de automóvil, trasladar materiales, etc. En una fábrica sin robots, los trabajos antes mencionados los realizan técnicos especialistas en cadenas de producción. Con los robots, el técnico puede librarse de la rutina y el riesgo que sus labores comportan, con lo que la empresa gana en rapidez, calidad y precisión.

En los próximos cien años, seguramente en todas las fábricas del mundo encontraremos robots trabajando.

Robots industriales

Los robots industriales se usan en su mayoría en los procesos de fabricación. Muchos de ellos son máquinas capaces de movimientos muy simples, originados por impulsos eléctricos o neumáticos. Gracias a estos movimientos se ejecuta una serie de operaciones previamente programadas.

ROBOTS IMPULSADOS NEUMÁTICAMENTE

La programación de estos robots consiste en la conexión de tubos de plástico a unos manguitos de unión de la unidad de control neumático. Esta unidad está formada por dos partes: una superior y una inferior. La parte inferior es un secuenciador que proporciona presión y vacío al conjunto de manguitos de unión en una secuencia controlada por el tiempo. La parte superior es el conjunto de manguitos de unión que activan cada una de las piezas móviles del robot. Las conexiones entre manguitos determinan qué piezas intervendrán en el movimiento, en qué dirección se moverán y los diferentes pasos que deberán efectuar. Modificando las conexiones de los manguitos de unión se podrán programar otras secuencias de pasos distintas.

Los robots del tipo descrito son los más simples que existen. Hay quien opina que a este tipo de máquinas no se les debería llamar robots; sin embargo, en ellas se encuentran todos los elementos básicos de un robot: estas máquinas son programables, automáticas y pueden realizar gran variedad de movimientos.

ROBOTS EQUIPADOS CON SERVOMECANISMOS

Otro tipo de robots más sofisticados desde el punto de vista del control y de las prestaciones que ofrecen son los que llevan servomecanismos.

El uso de servomecanismos va ligado al uso de sensores, como los potenciómetros, que informan de la posición del brazo o la pieza que se ha movido del robot, una vez éste ha ejecutado una orden transmitida. Esta posición es comparada con la que realmente debería adoptar el brazo o la pieza después de la ejecución de la orden; si no es la misma, se efectúa un movimiento más hasta llegar a la posición indicada.

ROBOTS PUNTO A PUNTO

Añadiendo a los servomecanismos una memoria electrónica capaz de almacenar programas y un conjunto de circuitos de control digital, se obtienen robots más potentes y de más fácil manejo.

La programación de este tercer tipo de robots se efectúa mediante una caja de control que posee un botón de control de velocidad, mediante el cual se puede ordenar al robot la ejecución de los movimientos paso a paso. Se clasifican, por orden de ejecución, los pasos que el robot debe seguir, al mismo tiempo que se puede ir grabando en la memoria la posición de cada paso. Éste será el programa que el robot ejecutará. Una vez terminada la programación, el robot inicia su trabajo según las instruciones del programa. A este tipo de robots se les llama *punto a punto*, porque el camino trazado para la realización de su trabajo está definido por pocos puntos. Para ejemplificar este método de programación pensemos en un niño que dirige un automóvil por control remoto. Si el vehículo dirigido tuviera una memoria que grabase los movimientos que el niño le ordena, podría realizar los mismos movimientos sin control y ser dirigido por la circuitería electrónica que ejecutaría el programa grabado en memoria.

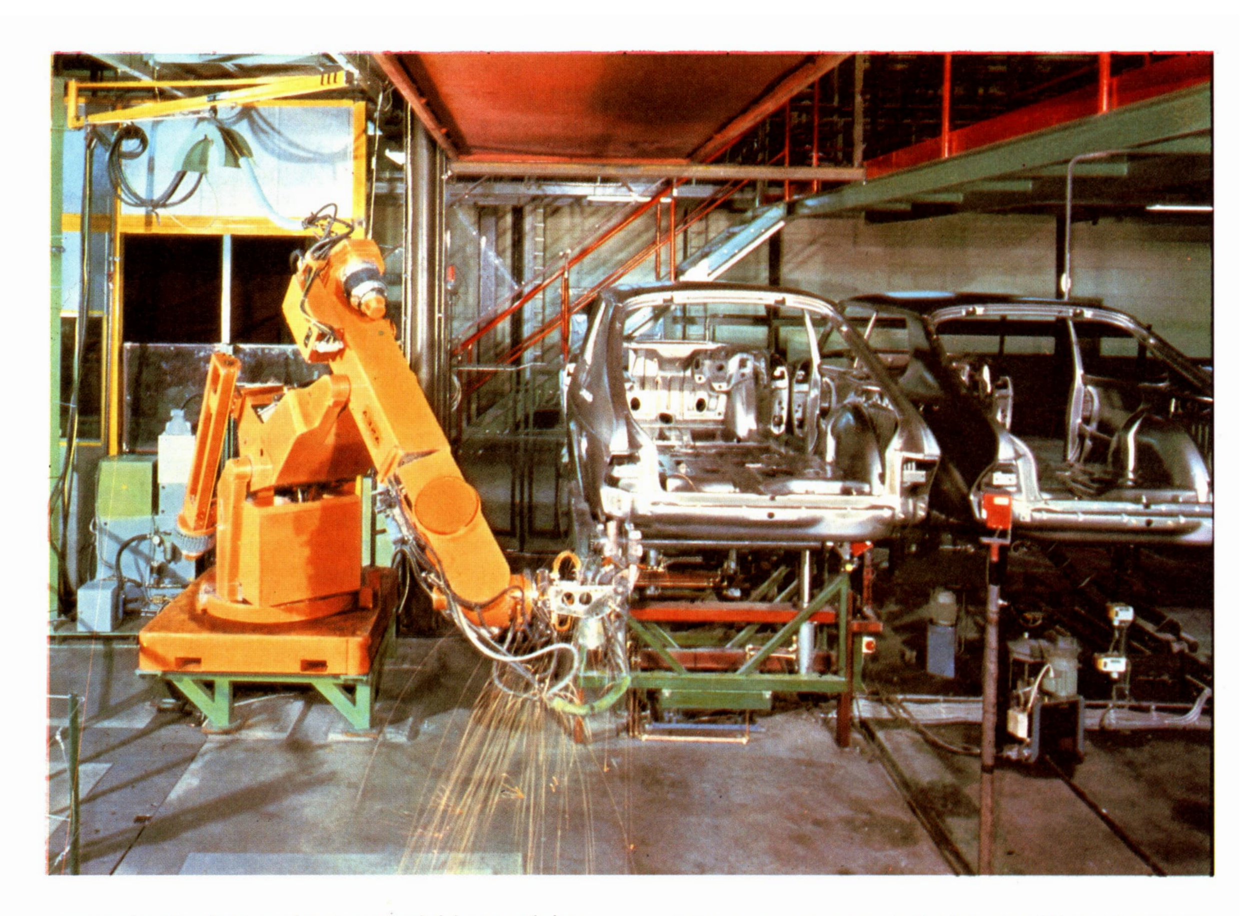
Gracias a la memoria electrónica que poseen estos robots, se pueden tener almacenados varios programas. El modo de elegir uno de los programas almacenados se hace a través de los recogidos por algún sensor o por una señal de input que les llega a través de las órdenes dadas por el programador.

Estos robots se usan por ejemplo en las cadenas de soldadura de carrocerías de automóviles. Los robots están programados para soldar automóviles de varios modelos distintos. El programador, o un sensor, reconoce el tipo de automóvil y decide el programa que se ha de aplicar en cada caso.

Estos programas constan de pocos pasos, muchas veces sólo cien; esto significa que no sirven como controladores de robots para trabajos de continuo movimiento. Para solventar este inconveniente, se usa una cinta en la que se almacenan miles de pasos de programa que el robot leerá y ejecutará; en estos casos la cinta actúa de memoria. Robots de este tipo, que se pueden encontrar en cadenas de pintura por spray, ya empiezan a trabajar como si fueran computadoras propiamente dichas.

ROBOTS CONTROLADOS POR COMPUTADORA

Un cuarto tipo de robots comprende aquellos que se pueden controlar mediante computadora. Con ella es posible programar el robot para que mueva sus brazos en línea recta o describiendo cualquier otra figura geométrica entre puntos preestablecidos. La programación se realiza mediante una caja de control o mediante el teclado de la computadora. El movimiento de sus brazos se especifica mediante varios sistemas de coordenadas según la referencia que se tome: la mesa de trabajo en la que se encuentra



Robot punto a punto trabajando en una fábrica de automóviles, soldando carrocerías.

apoyado el robot o el extremo del brazo del robot. La computadora permite además acelerar más o menos los movimientos del robot, para facilitar la manipulación de objetos pesados.

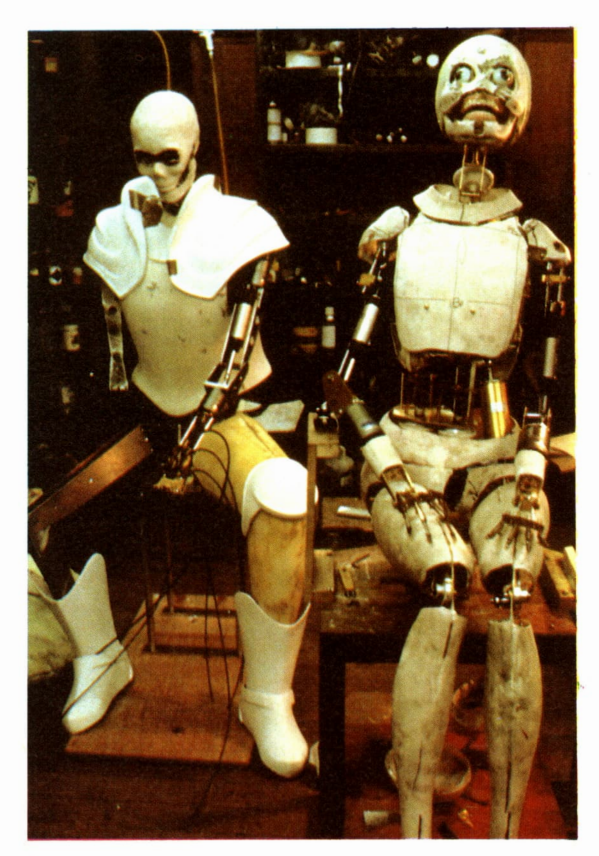
ROBOTS CON CAPACIDADES SENSORIALES

Aún se pueden añadir a este tipo de robots capacidades sensoriales: sensores ópticos, codificadores, etc. Los que no poseen estas capacidades sólo pueden trabajar en ambientes donde los objetos que se manipulan se mantienen siempre en la misma posición. En el caso de la cadena de soldadura de carrocerías de automóviles, las carrocerías están en movimiento hasta que llegan delante del robot, donde quedan inmóviles hasta que éste termina su trabajo; en este momento la cadena se vuelve a poner en movimiento hasta que vuelve a detenerse cuando otra carrocería está delante del robot, y así sucesivamente. Si estos robots tuvieran capacidades sensoriales, podrían suprimirse las paradas en la cadena. Supongamos que hay un codificador sujeto a la línea de movimiento y que el robot está provisto de un sensor óptico. El primero indicará al robot la velocidad de la carrocería y con el segundo el robot sabrá cuándo esta carrocería se mueve en su área de trabajo, momento en que empezará a ejecutar las órdenes que le llegan de la computadora. A partir de este momento, la computadora del robot irá transformando el sistema de coordenadas con respecto a la carrocería en movimiento para que el robot pueda efectuar las soldaduras en el lugar apropiado.

Los robots con capacidades sensoriales constituyen la última generación de este tipo de máquinas. El uso de estos robots en los ambientes industriales es muy escaso debido a su elevado coste. Actualmente, las compañías industriales están valorando si económicamente les resulta más ventajoso mantener los robots que necesitan tener inmóviles los objetos o bien este último tipo de robots. La razón del encarecimiento de estas máquinas es el alto coste de los aparatos sensoriales y del software utilizado para el manejo.

A pesar de todo, la investigación sobre los aparatos sensoriales está en pleno apogeo, lo que conducirá seguramente a un abaratamiento de éstos y a un aumento de su potencia y de sus capacidades.

Estos robots se usan en cadenas de embotellado para comprobar si las botellas están llenas o si la etiqueta está bien colocada.



La construcción de robots de personajes conocidos, como Marylin Monroe y el presidente Kennedy, puede contribuir a atraer la atención del gran público hacia las posibilidades de esta tecnología.

Futuro de la robótica

A pesar de que existen muchos robots que efectúan trabajos industriales, aquéllos son incapaces de desarrollar la mayoría de operaciones que la industria requiere. Al no disponer de unas capacidades sensoriales bien desarrolladas, el robot es incapaz de realizar tareas que dependen del resultado de otra anterior.

En un futuro próximo, la robótica puede experimentar un avance espectacular con las cámaras de televisión (ejemplo de aparato sensorial), más pequeñas y menos caras, y con las computadoras más potentes y más asequibles. Los sensores se diseñarán de modo que puedan medir el espacio tridimensional que rodea al robot, así como reconocer y medir la posición y la orientación de los objetos y sus relaciones con el espacio. Se dispondrá de un sistema de proceso sensorial capaz de analizar e interpretar los datos generados por los sensores, así como de compararlos con un modelo para detectar los errores que se puedan producir. Finalmente, habrá un sistema de control que podrá aceptar comandos de alto nivel y convertirlos en órdenes, que serán ejecutadas por el robot para realizar tareas enormemente sofisticadas.

Si los elementos del robot son cada vez más potentes, también tendrán que serlo los programas que los controlen a través de la computadora. Si los programas son más complejos, la computadora deberá ser más potente y cumplir unos requisitos mínimos para dar una respuesta rápida a la información que le llegue a través de los sensores del robot.

Paralelo al avance de los robots industriales será el avance de las investigaciones de los robots llamados *androides,* que también se beneficiarán de los nuevos logros en el campo de los aparatos sensoriales. De todas formas, es posible que pasen decenas de años antes de que se vea un androide con mínima apariencia humana en cuanto a movimientos y comportamiento.

LAS COMPUTADORAS Y EL ESPACIO

Otro de los campos en el que las computadoras son también un elemento fundamental es el relativo al estudio del espacio, sobre todo en dos materias principales: la astronomía y la exploración espacial. Al comparar estas dos materias, se deduce que la segunda ha llegado como consecuencia de la primera, por lo que los avances que se consigan en una de ellas redundarán forzosamente en la otra.

El uso de la computadora está mucho más extendido en la exploración espacial que en la astronomía propiamente dicha; tanto es así que no hubiera sido posible llegar a la Luna sin la ayuda de las computadoras.

A continuación, se describe el uso actual y futuro de las computadoras en astronomía.

Astronomía

Es una de las ciencias más antiguas que, gracias a las computadoras, ha experimentado un impacto revolucionario desde hace pocos años; concretamente en los observatorios más pequeños, en el área de la astronomía óptica.

Normalmente, en los grandes observatorios se trabaja desde hace mucho tiempo con computadoras de gran tamaño, gracias a que estos observatorios disponen de las instalaciones y el presupuesto que exigen estas máquinas.

Desde el momento en que la computación entró en el campo de la astronomía se han ido creando aplicaciones para realizar tareas astronómicas tradicionales, como son: conversión del calendario gregoriano al calendario juliano, conversión de la hora civil a la hora sideral, determinación de las horas de salida y puestas del Sol y la Luna respectivamente, etc. A partir de estas aplicaciones y gracias a ellas, han surgido una serie de publicaciones científicas en las que

aparecen muchos de los trabajos que dentro de este campo se realizan con la ayuda de la computadora; incluso se han difundido las fórmulas introducidas en una microcomputadora, que predicen con mucha precisión e instantáneamente la posición de los planetas. La determinación de las ecuaciones orbitales de los asteroides y cometas a partir de unas pocas observaciones siempre ha sido una labor muy difícil y delicada; sin embargo, con la colaboración de las microcomputadoras este cálculo se ha visto facilitado y dotado de mayor precisión y fiabilidad.

Hasta ahora sólo hemos hablado de las tareas de cálculo que realizan las computadoras para ayudar al astrónomo, pero también son útiles y se emplean para la reducción y el análisis de observaciones astronómicas. Estas observaciones pueden hacerse con telescopios muy pequeños equipados con fotómetros; esto quiere decir que no son necesarios los observatorios para realizar estas tomas ni tampoco para su análisis y reducción posterior. La entrada de las microcomputadoras ha facilitado en este campo el conocimiento y la difusión.

Es importante destacar la ayuda que representan las computadoras debido a la gran cantidad de datos que pueden almacenar referentes a objetos astronómicos (estrellas, galaxias, planetas, etc...) y a la facilidad con que pueden poner dichos datos al alcance de todo aquel que esté interesado en ellos. Así, por ejemplo, existe ya un catálogo de información acerca del brillo y la posición de más de 300.000 estrellas.

Los catálogos de información astronómica se encuentran en forma computerizada, es decir, los datos están almacenados en un dispositivo de almacenamiento externo de memoria: por ejemplo, una cinta magnética. En Estados Unidos, muchos de estos catálogos están depositados en el Centro de Datos Astronómicos, mientras que en Francia se encuentran en el Centro de Datos Estelares de Estrasburgo.

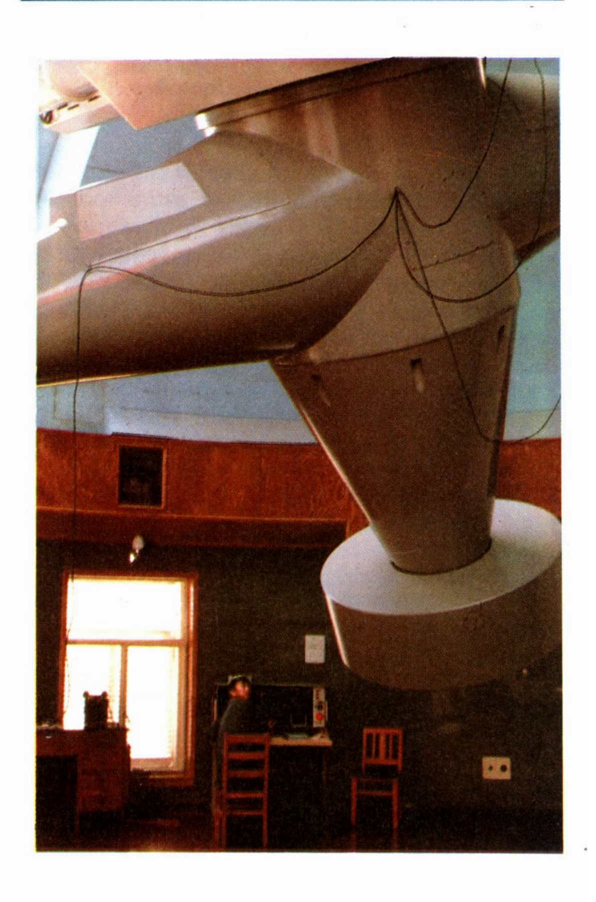
Además de los catálogos, se han confeccionado atlas, que son esencialmente mapas de estrellas elaborados a partir de la información que se encuentra en los catálogos. Estos atlas se han dibujado sobre largas piezas de papel con la ayuda de un plotter. En algunos de ellos está representada toda una galaxia.

Futuro de las computadoras en la astronomía

El futuro de las computadoras en el campo de la astronomía pasa por los primeros intentos para englobar dentro de una misma computado-

Telescopios controlados por computadora

Otra de las aplicaciones de las computadoras en la astronomía es el control de motores paso a paso para mover pequeños telescopios mediante dos ejes ortogonales. Uno de estos ejes tiene que ser paralelo al eje de la Tierra, para que pueda compensar su movimiento rotatorio, y, además, tiene que ser también a su vez rotatorio. Si se empieza a mover el telescopio a baja velocidad y ésta se va incrementando continuamente, se puede conseguir una velocidad muy elevada que permite los desplazamientos en amplios movimientos de observación del espacio. Para parar el movimiento puede seguirse la misma técnica disminuyendo la velocidad hasta conseguir una parada suave allí donde se desee. Dada la distancia angular que el telescopio debe recorrer entre dos objetos que se quieran observar, se pueden calcular exactamente los pasos que serán necesarios para realizar tal movimiento y cómo ejecutar el aumento o la disminución de la velocidad



Telescopio accionado por un motor paso a paso controlado por una computadora, situado en el observatorio astronómico del centro astronaútico de China Kunming.

ra todas las aplicaciones que hoy en día se están realizando, a saber:

• Catálogo de objetos astronómicos.
• Selección mediante la computadora de las estrellas que se quieren observar.
• Control del movimiento del telescopio.
• Control fotométrico de las estrellas para poderlas encontrar y medir.
• Análisis y reducción de datos de las observaciones.

Todo ello se podrá conseguir con la inserción de una placa dentro de la computadora.

Otro avance posible será el envío telefónico de listas de objetos que se quieran observar. Las observaciones requeridas se efectuarán automáticamente y los resultados de ellas se enviarán en sentido contrario al astrónomo o al solicitante de la información.

Por último, existen actualmente algunos astrónomos que manifiestan preocupaciones y problemas a causa del estado de la atmósfera. Para solucionarlos se están previendo estaciones espaciales dotadas de telescopios y controladas totalmente por microcomputadoras. De esta forma se resolverán los problemas del astrónomo respecto a la atmósfera, se tendrá acceso a mucha más información y se podrán efectuar observaciones desde otro lugar del espacio, que serán enviadas instantáneamente a las estaciones receptoras de la Tierra.

En el sistema PBX, la comunicación entre las diversas computadoras y periféricos que integran la red local se efectúa a través de una central telefónica privada, que funciona únicamente con dispositivos electrónicos y que opera sobre la base de la división de tiempo.

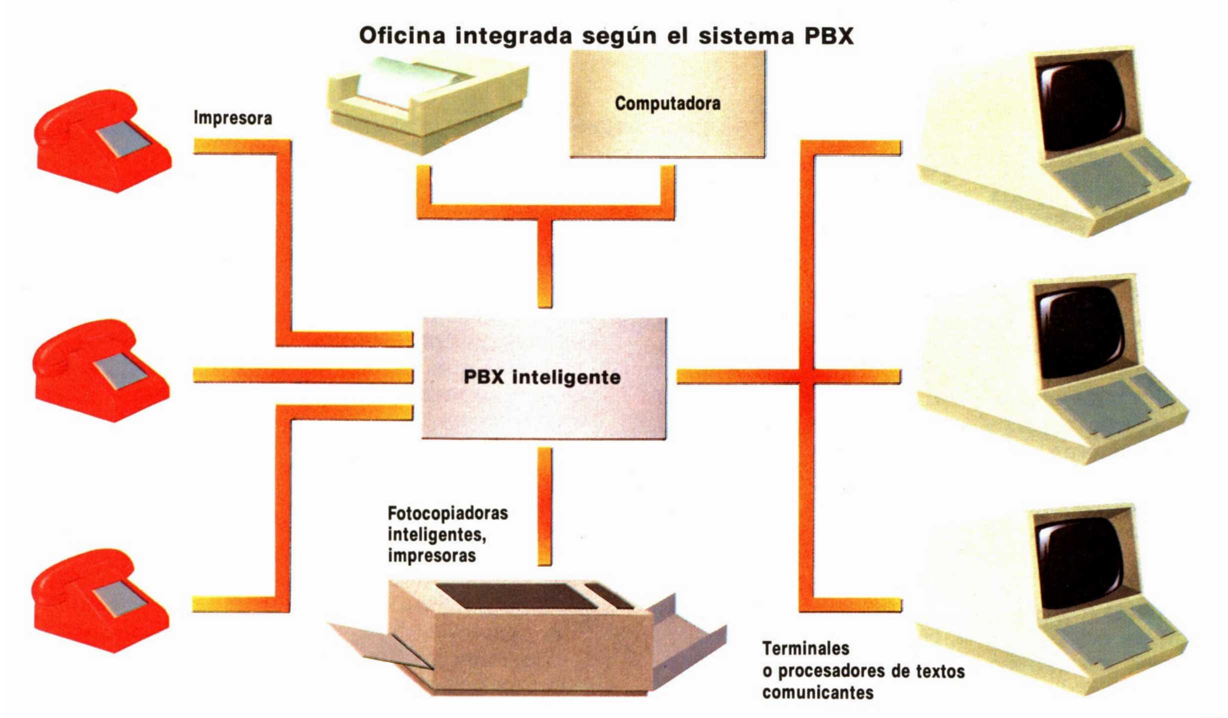
REDES LOCALES

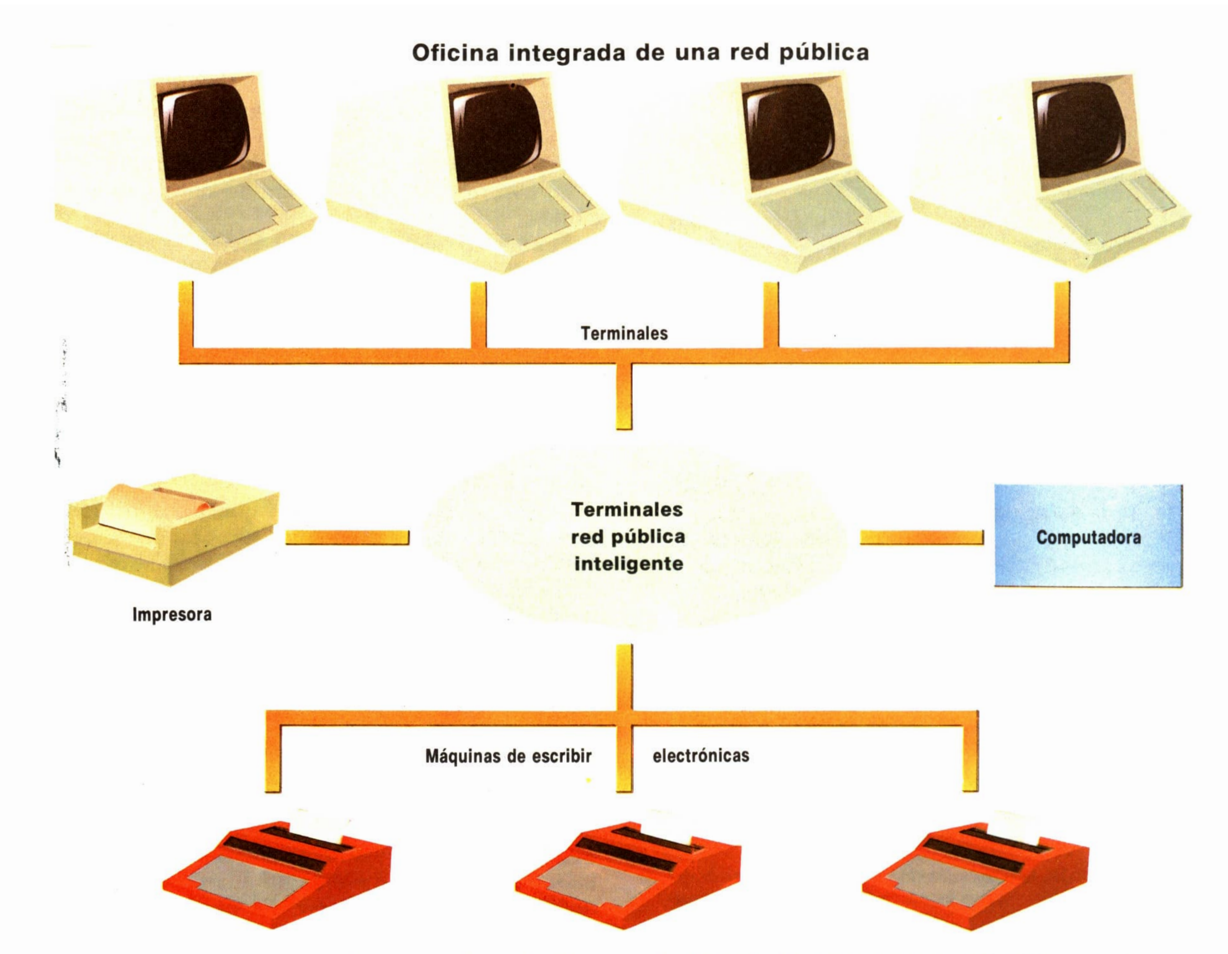
Las redes locales constituyen uno de los últimos avances tecnológicos en comunicaciones gracias a los progresos experimentados por el hardware y por el software de telecomunicaciones.

Éstas consisten en la conexión entre sí de diversos aparatos inteligentes (entre ellos las computadoras) para realizar intercambio de información y poder distribuirse mejor los recursos de trabajo. Dentro de una red local podemos encontrar por ejemplo un equipo trabajando con un proceso de textos, otro con una base de datos, otro con el correo electrónico, etc.; toda la información que estos equipos procesan es accesible a todos ellos.

Hace ya bastante tiempo que existen redes telefónicas para la comunicación oral entre personas, mientras que hace mucho menos que se han desarrollado las redes de datos para comunicación entre terminales de computadoras, computadoras y sistemas remotos. Las redes locales forman parte de estos últimos logros; a pesar de que la comunicación sea sólo a nivel local, la necesidad de comunicación es la misma que si fuera a otro nivel.

Cuando hablamos de comunicación local, nos referimos a la comunicación que se establece entre distancias no superiores a los 3 km. Supongamos, por ejemplo, que las distintas facultades de una universidad tienen necesidad de compartir información, ya sea de carácter técnico, científico o incluso administrativo. Estos intercambios de información son puramente





En las redes de dimensiones considerables, acostumbra a emplearse la red telefónica para la interconexión de los diversos órganos que la integran.

locales; sin embargo, son tanto o más necesarios que las comunicaciones que puedan establecerse entre universidades distintas.

Necesidad de las comunicaciones locales

Las oficinas de las grandes empresas, e incluso de las más pequeñas, están cada día más mecanizadas, por lo que no es raro encontrar en ellas máquinas cada día más potentes e inteligentes que realizan casi todo el trabajo diario; pero sí es raro que el intercambio de información entre los diferentes departamentos de la compañía se realice a través de estas máquinas computadoras.

En cualquier oficina, se pueden hallar computadoras personales, terminales inteligentes, procesadores de textos, impresoras, copiadoras, teléfonos y plotters. Si trabajando con estas máquinas de manera individual se han conseguido unos niveles de eficacia en el trabajo de oficina muy importantes, se puede conseguir mucho más integrando todas estas máquinas en redes locales. Estas redes permiten que todos los elementos conectados utilicen unos recursos insospechados hasta el momento. Gracias a la red local, las máquinas pueden comunicarse con velocidad, rapidez y seguridad. Los bancos de datos, los programas, los ficheros y demás recursos computacionales de una compañía se pueden construir, mantener y mejorar normalmente desde cualquier equipo conectado a la red local, siempre que la potencia se lo permita (una computadora o un terminal). La información está más distribuida y ya no es propiedad de un solo departamento o individuo. Las personas que utilizan las computadoras, terminales y procesadores de textos pueden realizar intercambios de datos, enviar mensajes entre sí, acceder a bancos de datos comunes para poder preparar informes, estudios, ponencias o simplemente para consultar algunos datos, compartir aplicaciones para no caer en la redundancia de dos aplicaciones para un mismo trabajo y, por último, compartir los dispositivos de almacenamiento externo (discos duros, cintas, diskettes) y los dispositivos de entrada-salida (impresoras, plotters, copiadoras, scanners, lectores ópticos y otros).

Oficina integrada de computadoras centrales

Base de datos

Mini «esclavo»

Computadora

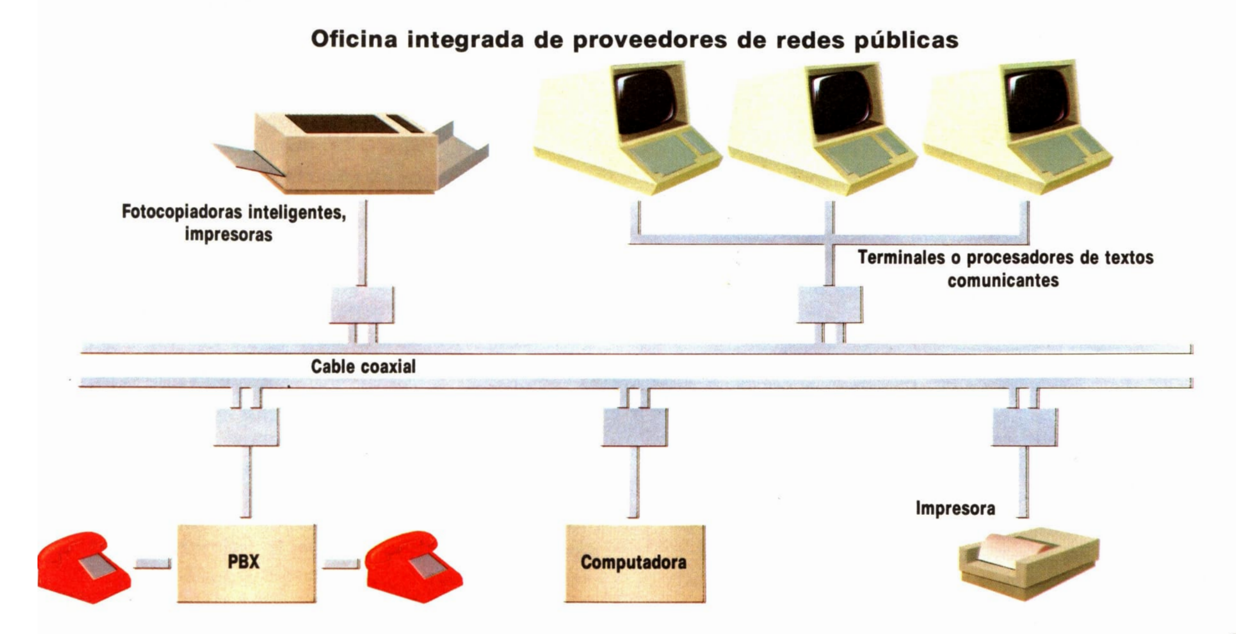
Terminales remotos

Terminales locales

Arriba. *En las redes concentradas, una computadora central de elevada capacidad está conectada directamente con las diversas terminales, tanto locales como remotas, y periféricos. Las informaciones están así centralizadas en un sólo punto.*
Abajo. *Las redes públicas permiten cubrir áreas geográficas muy amplias, y su desarrollo ha sido posible gracias a la estandarización de los métodos de acceso y a la utilización, por parte de la compañía telefónica, de computadoras destinadas a gestionar el tráfico de informaciones por la red.*

Tipos de redes

Al hablar de redes locales, hay dos términos que se emplean frecuentemente: *enlace* y *nodo*. Ambos son fundamentales en las redes locales, ya que el tipo de red depende de su disposición geométrica.



Un *nodo* es el punto de conexión de cualquier sistema inteligente a la red. Un *enlace* es el recorrido que hay entre un nodo y otro.

Normalmente, cuando nos referimos a un nodo hacemos alusión al sistema o computadora que se encuentra conectada a la red.

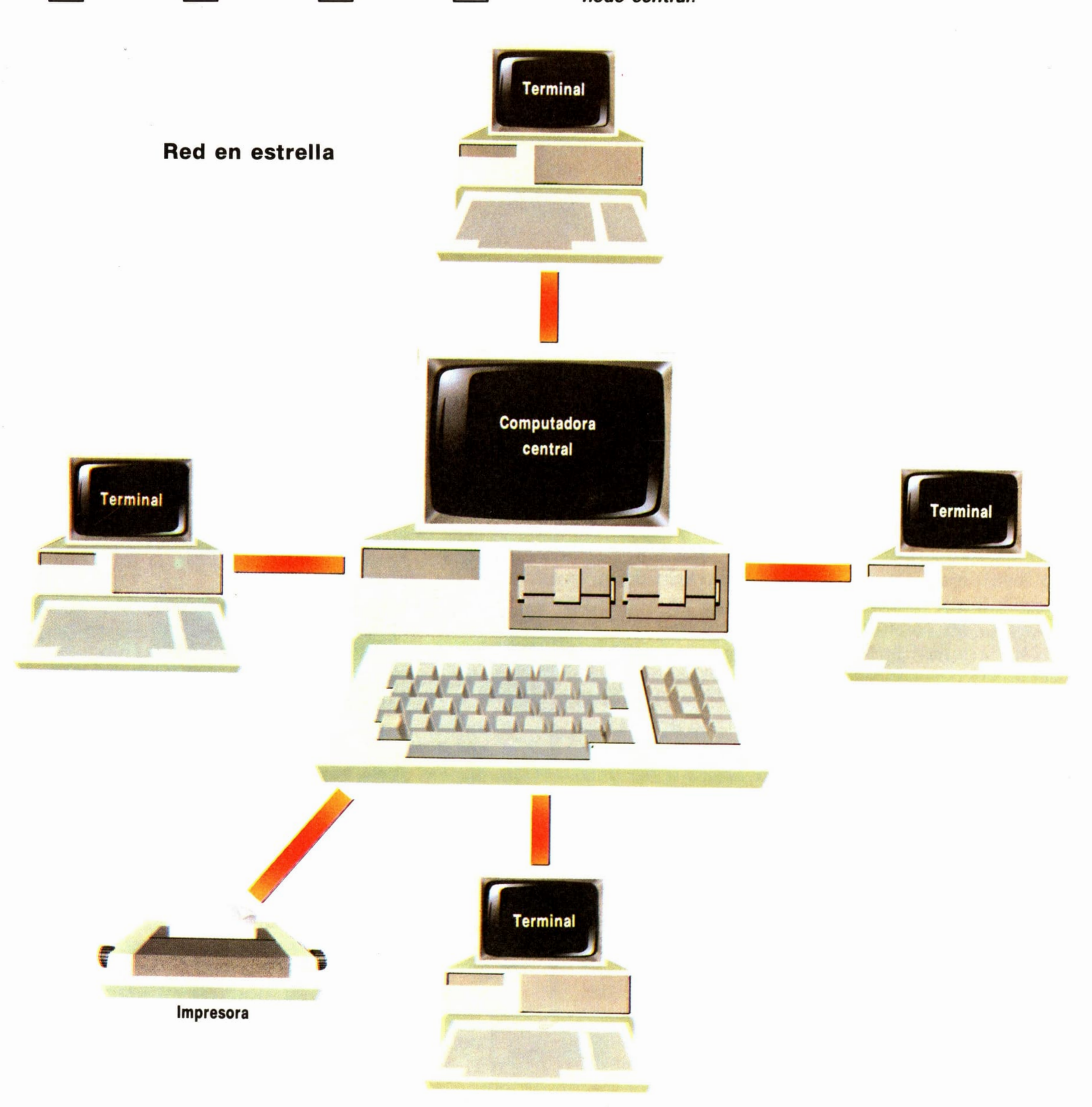
Los nodos se comunican entre sí dentro de la red a través de enlaces físicos y lógicos. Los enlaces físicos son cables eléctricos que unen los nodos de manera permanente o temporal. El enlace lógico se efectúa entre dos nodos sin necesidad de que estén unidos físicamente.

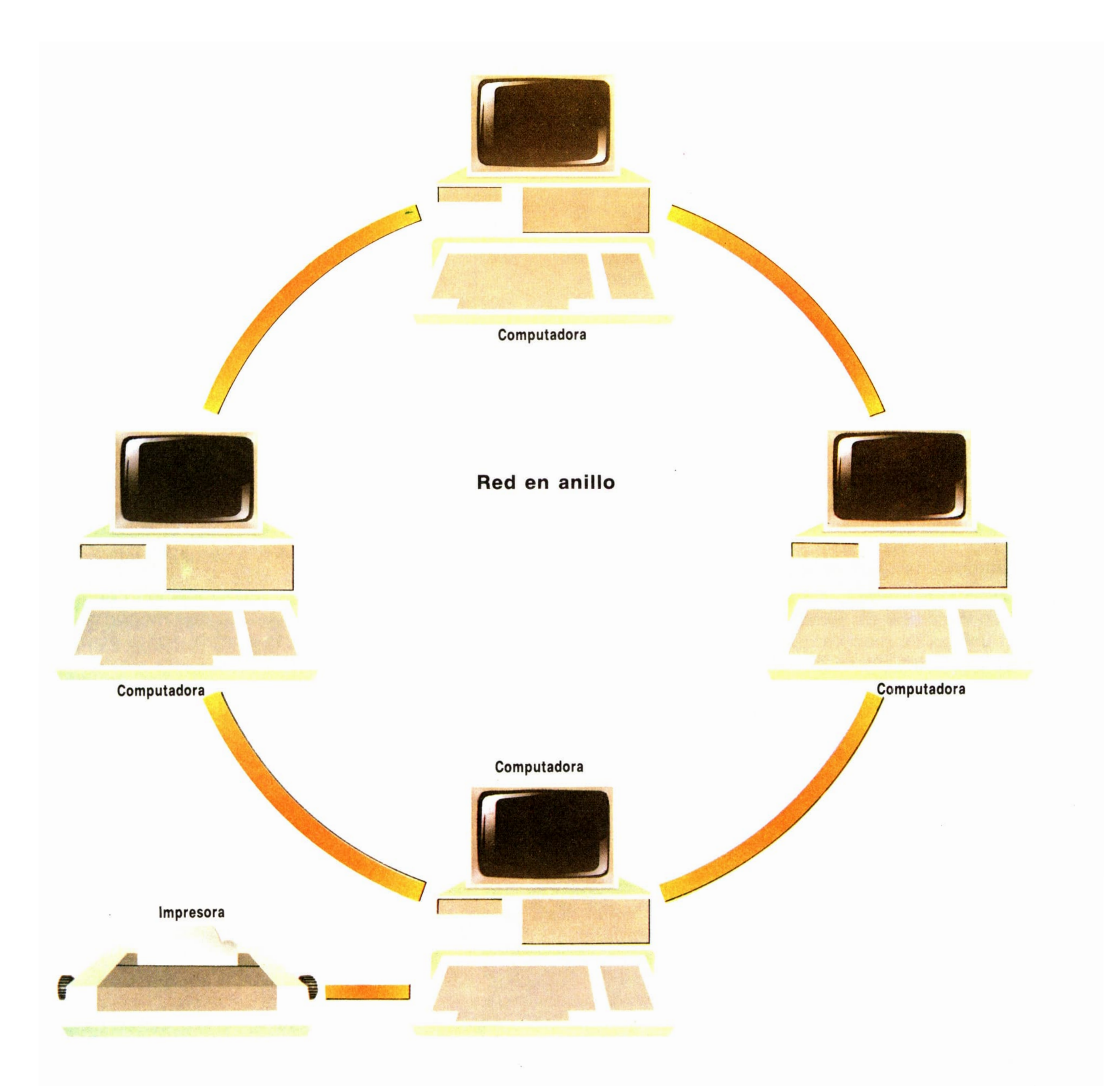
Supongamos la disposición geográfica de la siguiente red:

A ⟶ B ⟶ C ⟶ D

En esta disposición sólo están conectados directamente A con B, B con C y C con D. Si la comunicación fuera exclusivamente física, está sólo se podría establecer entre los elementos directamente conectados; es decir, sólo establecerían comunicación A con B, B con C y C con D. Pero como la comunicación es lógica, A puede comunicarse con D a voluntad; basta con enviar información a D desde A a través de B y C. A enviará el mensaje a B comunicándole que es para D y B hará lo mismo con C hasta que D reciba la información transmitida.

Red local con topología en estrella. La computadora central distribuye las informaciones a los nodos periféricos y la comunicación entre éstos ha de pasar necesariamente por el nodo central.





Red local con topología en anillo. La estructura de esta red no está supeditada a un sólo nodo: cada mensaje pasa a través de todas las estaciones, hasta que encuentra una que lo identifica como dirigido a ella y lo recoge.

Otro aspecto importante de los tipos de redes es el control de red. Este control puede estar centralizado o distribuido. En el caso de un control centralizado, uno de los nodos controla el acceso a la red (decide qué nodos pueden transmitir y cuándo) y la asignación del enlace (decide qué parte del enlace puede ser utilizada por un nodo y durante cuánto tiempo). En el caso de un control distribuido, cada nodo puede usar y transmitir el enlace con independencia de los demás. En algunas redes locales, el control del acceso está dividido en partes iguales, o sea, todos los nodos tienen las mismas posibilidades de utilizar la red para transmitir datos.

Según la disposición geográfica de los nodos y del control de la red, las redes locales se dividen en: redes en estrella, redes en anillo y redes en bus.

REDES EN ESTRELLA

En este tipo de redes todos los nodos están conectados a uno de ellos tal y como se ilustra en la figura **red en estrella** *(pág. 248).*

Según se efectué el control de estas redes se puede trabajar de tres maneras:

• El control de la red se asigna al nodo central, de tal modo que todos los mensajes que se envían son controlados por él.

Este tipo de redes es muy útil en ambientes de trabajo en los que se precisa que la información esté centralizada en un solo punto y para hallarla hay que acudir a dicho punto. Por el contrario, si la información se encuentra distribuida en varios lugares, este sistema no es el idóneo para realizar comunicaciones entre ellos.

• El control de la red se asigna a un nodo exterior.

• El control se encuentra distribuido de manera generalizada en todos los nodos exteriores.

En los últimos casos, la función de control del nodo central es únicamente la de un simple conmutador que establece los enlaces entre los nodos exteriores. El nodo central es el punto crítico en todas las redes en estrella. Cuando sufre una avería, toda la red queda inutilizada.

La potencia de este tipo de redes depende de la potencia de su nodo central. Cuanto más potente sea la computadora que hace de nodo central más potencia tendrá la red.

REDES EN ANILLO

En este tipo de redes los nodos se encuentran unidos de manera que forman una configuración circular sin ninguna interrupción, tal como muestra la figura **red en anillo** *(pág. 249)*.

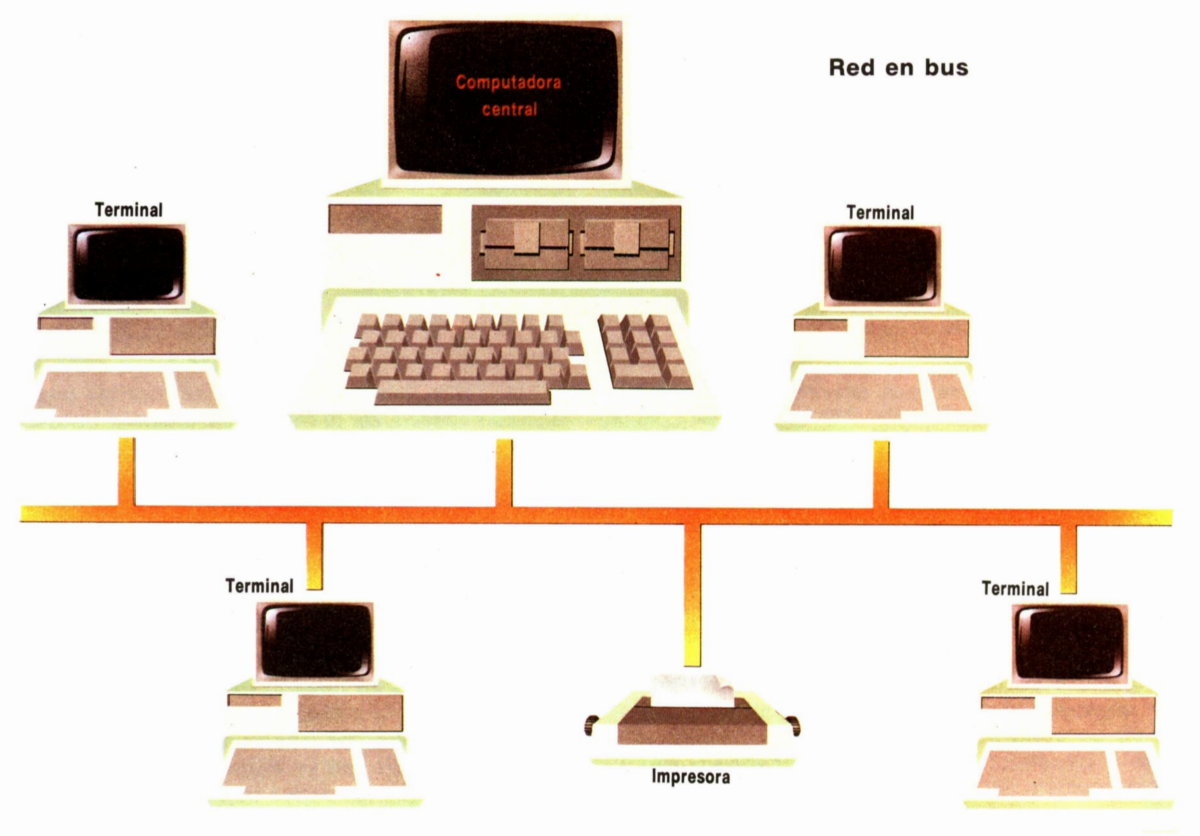
La información viaja a través de los nodos con el identificativo del nodo a la que va destinada. Los nodos tienen la posibilidad de reconocer el identificativo; en el caso de que éste coincida con el que lleva la información, el nodo correspondiente recoge el mensaje; si no ocurre así, es decir, si no coincide, lo envía al nodo siguiente.

Éste es un ejemplo de red distribuida, ya que todos los nodos tienen las mismas posibilidades de transmisión.

El sistema que se usa en este tipo de red para realizar las transmisiones es un poco más sofisticado, debido a que no hay un solo nodo que las controle, sino que todos efectúan el control. El sistema de transmisión se basa en una señal que va recorriendo todo el anillo; cuando esta señal es recogida por un nodo, éste es el que puede transmitir y ninguno más. Cuando el nodo que ha recogido la señal ha dejado de transmitir, esta señal queda libre para que otro pueda recogerla y efectuar la transmisión.

Esta red funciona a través de un enlace común en el que todos los nodos están conectados de manera lógica, y cada uno de ellos puede conectar o establecer comunicación con los demás por iniciativa propia, sin que ningún otro nodo ejerza algún tipo de control sobre él.

Red local con topología en bus. Esta configuración comporta un único canal central o* bus*, al que pueden acceder directamente todas las estaciones, transmitiendo información y recogiendo la que vaya destinada a ellas.



Debido a la forma circular y cerrada de la red, cabría pensar que la información puede viajar en cualquier sentido. Pero no ocurre así; la información únicamente puede circular en un solo sentido: el de las agujas del reloj. Como consecuencia, cualquier avería en un nodo de los que forman la red, la inutiliza toda.

REDES EN BUS

Los nodos van conectados a una línea totalmente abierta, de tal manera que se puede ir ampliando por simple conexión de nodos en sus extremos (véase **red en bus** *[pág. 301]*).

El término *bus* se refiere a la línea de transmisión común a todos los nodos.

La información transmitida por cualquier nodo viaja a través del bus con el identificativo del nodo al cual va dirigida, y el proceso de recogida de la información se efectúa del mismo modo que en las redes en anillo. Si el nodo reconoce el identificativo y coincide con el suyo, recoge el mensaje; en caso contrario, el mensaje sigue recorriendo el bus. Con este sistema ya no es preciso que los nodos manden mensajes al nodo siguiente, ya que la información va circulando por el bus. Así se gana el tiempo que, en una red en anillo, se pierde normalmente al tener que enviar los mensajes a los nodos a los que no van dirigidos.

En este tipo de red todos los nodos tienen las mismas posibilidades de transmisión. La avería de un nodo no afecta, pues, al funcionamiento general de toda la red, ya que ésta sigue funcionando, sólo que con un nodo averiado, como si éste estuviera desconectado.

La red en bus es el tipo de red local más utilizado debido a su bajo coste, su sencilla instalación y la poca complejidad de las tecnologías de transmisón que utiliza.

Medios de transmisión utilizados por las redes

Los medios de transmisión proporcionan el enlace físico entre los diversos nodos de una red. Se clasifican en:

- *medios ligados* (cables y fibras ópticas);
- *medios no ligados* (ondas electromagnéticas, microondas).

CABLES

Los cables constituyen el medio de transmisión más usual; los hay de dos tipos: el par y el cable coaxial.

• El *par* se utiliza para transmisiones locales de telefonía y de datos. Los pares de cables trenzados se enrollan juntos para reducir interferencias. Es el tipo de cable menos costoso que se puede utilizar como medio de transmisión de una red local; sin embargo, tiene el inconveniente de que puede absorber muchas interferencias eléctricas producidas por otros aparatos. Este inconveniente hace necesario aislar la red de posibles interferencias que reducían el nivel de seguridad de la información que viaja por ella.

• El *cable coaxial* puede soportar la transmisión de gran cantidad de datos al mismo tiempo. Presenta una buena protección contra las interferencias eléctricas y contra los errores espontáneos.

Se utiliza ampliamente en la red telefónica para poder transmitir muchas llamadas por un solo cable, eliminando así la instalación de millares de hilos simples.

Este tipo de cable está formado por un hilo conductor simple, recubierto por un aislamiento, que va alojado en el interior de un manguito de aluminio extruido o de una malla de hilo de cobre.

Este cable se ha adaptado para su uso en redes locales por su potencia, su protección contra los errores y su gran aislamiento. Se emplea sobre todo en las redes de bus multipunto.

FIBRAS ÓPTICAS

De reciente creación, las fibras ópticas están destinadas en un futuro próximo a ser el medio de transmisión por excelencia, gracias a su extraordinaria potencia de transmisión.

Están hechas de plástico o de vidrio y pueden proporcionar soporte a cantidad de datos superiores a 1 Gbit por segundo (mil millones de bits por segundo).

La probabilidad de error es muy baja: un bit por cada mil millones transmitidos. La transmisión no está sujeta a interferencias eléctricas o electromagnéticas; además, las fibras ópticas no producen perturbaciones. Por estos motivos son el medio de transmisión más seguro hasta ahora. También tienen la ventaja de ser muy pequeñas y ligeras, con lo que se ahorra espacio y peso. Unas cuantas fibras ópticas pueden transportar la señal de centenares de cables trenzados.

En la transmisión, las señales eléctricas se convierten en impulsos luminosos por medio de un modulador; éstos viajan por la fibra óptica desde una fuente luminosa, se reciben y se reconvierten en señales eléctricas por medio de diodos fotoeléctricos. Estas fibras están revestidas de un material opaco que impide la pérdida de señal luminosa y la entrada de luz exterior. Las señales luminosas, o la luz que viaja por las fibras, pueden transmitirse en forma analógica o digital. Las señales luminosas de tipo analógico varían en función de la intensidad de la luz, mientras que las señales luminosas digitales son impulsos luminosos del tipo encendido-apagado.

ONDAS ELECTROMAGNÉTICAS Y MICROONDAS

Cualquier transmisión puede efectuarse sin ningún tipo de hilo, simplemente mediante el éter. Éste es el caso de gran parte de las transmisiones de información que se reciben a diario, por la radio y por la televisión. Este tipo de transmisión utiliza ondas electromagnéticas o microondas, que se propagan por el éter.

Los tres tipos principales de señales que se transmiten por éter son: las señales de radio, las señales infrarrojas y las microondas.

Muy pocas redes locales se basan en un sistema de transmisión de este tipo, ya que todavía están en fase de experimentación.

El futuro de las redes locales

A pesar del conocimiento que actualmente se tiene sobre redes locales, hay todavía un largo camino por recorrer para llegar al objetivo de toda la red local: que cualquier máquina inteligente pueda conectarse a esta red sin que ello signifique un obstáculo que impida su buen funcionamiento.

Hoy por hoy, la computadoras, los procesadores de textos, las copiadoras, etc., son incompatibles a no ser que todas estas máquinas procedan del mismo fabricante. En el futuro se deberán dictar unas normas únicas de conexión para todos los fabricantes, de tal modo que se pueda montar una red local con cualquier tipo de aparato, sin necesidad de que sea del mismo fabricante y que ello no afecte al funcionamiento de la red.

Esto proporcionará a los usuarios una amplia gama donde elegir las máquinas, con lo que se obtendrán precios más ventajosos y mejorará la calidad de los productos.

Las redes locales, en el futuro, se podrán conectar a otras redes locales o incluso a las redes de larga distancia.

El futuro de los medios de transmisión de las redes locales son las fibras ópticas y los medios no ligados.

Transmisión potente, rápida y precisa: fibras ópticas

En la actualidad, el empleo de las fibras ópticas en las redes locales es muy limitada, ya que las tecnologías de transmisión y enlace construidas para dichas redes no están a la altura de aquéllas. Éste es el motivo por el cual las fibras ópticas todavía son menos eficientes que los cables. Cuando las técnicas de transmisión se hayan desarrollado de acuerdo con la potencia de este medio, no sólo se utilizarán masivamente en la conexión de redes locales, sino también en cualquier sistema donde sea necesaria una transmisión potente rápida y precisa. A pesar de este desfase entre la transmisión y su medio, hay algunos fabricantes que ya han adoptado las fibras ópticas, por ser éstas el medio de transmisión que da mejor resultado.

Fibras ópticas: Detalle de fibras individuales con emisión de luz polícroma.



ÍNDICE

plan general de la obra

VOLUMEN 1

VOLUMEN 2

Periféricos
Introducción. Tipos de periféricos
Unidades de impresión de datos.
Unidades de soporte de información. Plotters y digitalizadores
Módems y adaptadores de línea
Lectores ópticos de caracteres. Scanners. Periféricos de control de presencia

VOLUMEN 3

Software
Software de base. Software aplicativo
Sistema operativo
Conceptos básicos en la programación
Algoritmos y organigramas. Sistemas de numeración
Organización de los datos
Lenguajes de programación
Los lenguajes assembler. Los lenguajes de alto nivel
El lenguaje BASIC
El futuro tecnológico científico y las computadoras
La robótica. Las computadoras y el espacio
Redes locales

VOLUMEN 4

La computadora: una máquina para enseñar y aprender
La computadora y la educación. La alfabetización computacional
Alfabetización funcional
Imagen y texto
Programas educativos
Criterio para conocer los programas. Reforzamiento de estructuras
Simulación. Otras técnicas computacionales
Programas actuales y programas para el futuro
La inteligencia artificial y la quinta generación
Un futuro difícil de predecir
Prototipos hoy, realidad cotidiana mañana. ¿Puede pensar una máquina?
La inteligencia artificial
La quinta generación de computadoras
Instrucciones más comunes del lenguaje BASIC
Instrucciones de definición de matrices. Instrucciones de asignación
Instrucciones de entrada de datos. Instrucciones de salida de datos
Instrucciones de control. Funciones del BASIC
Instrucciones de manejo de ficheros en disco. Instrucciones gráficas
Algunos ejemplos de programas escritos en lenguaje BASIC
Glosario. Diccionario inglés-español

OCEANO